Optimal-Generation Variants of EPEA*

Meir Goldenberg, Ariel Felner
ISE Department
Ben-Gurion University
Israel
mgoldenbe@gmail.com, felner@bgu.ac.il

Nathan Sturtevant
Computer Science Department
University of Denver
USA
Sturtevant@cs.du.edu

Robert C. Holte, Jonathan Schaeffer
Computer Science Department
University of Alberta
Canada
rholte@ualberta.ca, jonathan@cs.ulaberta.ca

Abstract

It is known that A* is optimal with respect to the *expanded* nodes (Dechter and Pearl 1985) (D&P). The exact meaning of this optimality varies depending on the class of algorithms and instances over which A* is claimed to be optimal. A* does not provide any optimality guarantees with respect to the generated nodes. However, such guarantees may be critical for optimally solving instances of domains with a large branching factor. In this paper, we introduce two new variants of the recently introduced Enhanced Partial Expansion A* algorithm (EPEA*) (Felner et al. 2012). We leverage the results of D&P to show that these variants possess optimality with respect to the generated nodes in much the same sense as A* possesses optimality with respect to the expanded nodes.

The results in this paper are theoretical. A study of the practical performance of the new variants is beyond the scope of this paper.

Introduction

A* and its derivatives such as IDA* (Korf 1985) and RBFS (Korf 1993) are general best-first search solvers guided by the cost function $f(n) = g(n) + h(n)$. Given an *admissible* (i.e. non-overestimating) heuristic function h, A* is guaranteed to find an optimal solution.

Until recently, performance studies of A* mostly focused on the number of nodes that A* expanded. Furthermore, a general result about the optimality of A* with respect to the expanded nodes has been established (Dechter and Pearl 1985) (D&P). However, the number of expanded nodes clearly does not represent the performance of A* accurately enough. Namely, let C^* be the cost of the optimal solution for a particular problem instance. A* has to expand all nodes n with $f(n) < C^*$ in order to guarantee optimality of the solution.[1] These nodes have to be generated as well. In addition, A* may generate many nodes with $f(n) \geq C^*$. The number of expansions of nodes with $f(n) = C^*$ depends on tie-breaking. The nodes with $f(n) > C^*$ are never expanded by A*. Following (Felner et al. 2012), we designate the nodes with $f(n) > C^*$ as *surplus*. Surplus nodes do not contribute to finding an optimal solution, so that generating them is a pure overhead. We designate the non-surplus nodes as *useful*.

Many important problem domains feature a large branching factor. When solving these problems, A* may generate a large number of surplus nodes. In fact, the execution time spent on generating these nodes may be much larger than the time spent on generating and expanding the useful nodes. When solving instances of such domains, the number of generated nodes is a central performance issue. Motivated by the need to address this issue, the recently introduced EPEA* algorithm (Felner et al. 2012) focuses on reducing the number of nodes that A* generates. EPEA* uses domain-specific knowledge to completely avoid the generation of surplus nodes. EPEA* was applied to a variety of domains with the result of improved performance.

The contributions of this paper are three-fold. First, we provide a concise and clear summary of the optimality results from the seminal work of D&P. Second (and most important), we introduce two new variants of EPEA* and leverage the results of D&P to show that these variants possess optimality with respect to the generated nodes in much the same sense as A* possesses optimality with respect to the expanded nodes. Third, we show that even some negative results from D&P (i.e. that, under some conditions, there does not exist an optimal algorithm with respect to the expanded nodes) can be directly translated into the realm of optimality with respect to the generated nodes.

In this paper, we are mostly interested in establishing that there exist algorithms that are optimal with respect to the generated nodes. Therefore, we discuss only the theoretical properties of the new variants of EPEA*. We leave it for future research to see whether practical benefits (i.e. reduction in running time) can be gained by solving difficult instances of domains with a large branching factor with these variants.

[1] In this statement, we assumed a consistent heuristic for simplicity. In general, A* has to expand all nodes n such that there is a path from the start to n, on which all nodes n' have $f(n') < C^*$.

Optimality of A* With Respect to the Expanded Nodes (D&P)

In order to give accurate statements of A*'s optimality, D&P introduced a number of classes of algorithms and problem instances, as well as four types of optimality over these classes. In this section, we cover these classes and the optimality results proven by D&P. As we stated in the introduction, we aim to both **(1)** provide a concise and clear description of the results of D&P and **(2)** point out the correct context of the much cited results about the optimality of A*.

The reader needs to have following two general points in mind in order to correctly interpret the results of D&P:

(1) D&P consider the heuristic as being part of the instance rather than part of the algorithm. Thus, a problem instance consists of the graph, the start and the goal states, and the heuristic.

(2) A* is defined as a *family* of best-first search algorithms, which expand nodes in the order defined by the additive combination $f(n) = g(n) + h(n)$. Each member of this family is A* with a particular tie-breaking rule. The only restriction on the possible tie-breaking rules is that the goal node is given priority over all other nodes with the same f-values.

Hereafter, when it is clear from the context whether we are talking about the A* family as a whole or about an arbitrary member of it, we will write simply A*. In particular, D&P established all of their optimality results for the A* family as a whole.

Classes of Algorithms

The A* family is a subset of each of the following classes of heuristic-driven search algorithms. It is assumed that all these algorithms operate by expanding nodes to generate their children. That is, a node cannot be expanded without having been generated first.

D&P explored the optimality of A* over each of the following classes:

(1) $\mathbf{A_{AD}}$ – admissible algorithms. These are the algorithms that are guaranteed to find an optimal solution for instances with an admissible (i.e. non-overestimating) heuristic.

(2) $\mathbf{A_{gc}}$ – algorithms that are *globally compatible* with A*. An admissible algorithm A is called *globally compatible* with A* if, given a non-admissible heuristic h, A finds an optimal solution for every instance that has this h and for which A* finds an optimal solution.

(3) $\mathbf{A_{bf}}$ – best-first search algorithms. These algorithms expand nodes in a best-first manner based on some cost function $f(n)$. They may perform any tie-breaking with one restriction: a goal node has precedence over all other nodes with the same cost. A* is a particular case of a best-first search algorithm, where $f(n)$ is an additive combination of $g(n)$ and $h(n)$.

In the following discussion, whenever we write "a class of algorithms", we will mean one of these three classes.

Classes of Instances

A problem instance consists of the graph, the start and the goal states, and the heuristic. D&P consider only instances with an admissible heuristic. They divide all such problem instances into two broad categories: *non-pathological* and *pathological*. A problem instance is called *non-pathological* if there exists an optimal cost solution path P such that, for all non-goal nodes n on P, the strict inequality $h(n) < h^*(n)$ holds, where h^* is a perfect heuristic. Otherwise, a problem instance is called *pathological*. In other words, a problem instance is non-pathological if there is an optimal cost solution path P such that $f(n)$ is below the cost of the optimal solution C^* for all nodes n on P except for the goal node. Note that there are problem domains and heuristics, for which all problem instances are pathological. For example, in the 15-puzzle, all instances with the Manhattan distance heuristic are pathological, since this heuristic is always perfect one move before the goal is reached.

D&P consider two types of heuristics: *consistent* and *inconsistent*. A heuristic is *consistent* if for every two states x and y, the inequality $h(x) \leq c(x, y) + h(y)$ holds, where $c(x, y)$ is the cost of the shortest path between x and y. Otherwise, the heuristic is called *inconsistent*.

D&P explored the optimality of A* with respect to expanded nodes over four classes of instances:

(1) $\mathbf{I_{AD}}$ – (possibly pathological) problem instances with an admissible (but possibly inconsistent) heuristic

(2) $\mathbf{I^{-}_{AD}}$ – non-pathological problem instances with an admissible (but possibly inconsistent) heuristic

(3) $\mathbf{I_{CON}}$ – (possibly pathological) problem instances with a consistent heuristic

(4) $\mathbf{I^{-}_{CON}}$ – non-pathological problem instances with a consistent heuristic

In the following discussion, whenever we write "a class of instances", we will mean one of these four classes. It is important to note that each class includes all possible input graphs. That is, for every input graph G, start state s and goal state g, depending on the heuristic h, the instance $I = (G, I, s, g)$ can be in any of the four classes of instances.

For an intermediate summary, there are three classes of algorithms and four classes of instances. The Cartesian product is shown in Table 1 (reproduced from D&P). This table has 12 cells. Each of these cells D&P shows two things: **(1)** A* is optimal in some sense and (if applicable) **(2)** there is no algorithm that is optimal in a stronger sense. D&P introduced four senses of optimality, which we cover next.

Types of optimality

D&P distinguished four types of optimality numbered from 0 to 3, as follows.

(0) A family of algorithms $\mathbf{B}$ (such as A* with all possible tie-breaking rules) is said to be 0-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $A \in \mathbf{A}$ and every problem instance $I \in \mathbf{I}$, every algorithm $B \in \mathbf{B}$ (in the case of A*, this is A* with some arbitrarily chosen tie-breaking rule) expands

	$\mathbf{A_{AD}}$	$\mathbf{A_{gc}}$	$\mathbf{A_{bf}}$
$\mathbf{I_{AD}}$	A** is 3-optimal no 2-optimal exists	A* is 1-optimal no 0-optimal exists	A* is 1-optimal no 0-optimal exists
$\mathbf{I^{-}_{AD}}$	A* is 2-optimal no 1-optimal exists	A* is 0-optimal	A* is 0-optimal
$\mathbf{I_{CON}}$	A* is 1-optimal no 0-optimal exists	A* is 1-optimal no 0-optimal exists	A* is 1-optimal no 0-optimal exists
$\mathbf{I^{-}_{CON}}$	A* is 0-optimal	A* is 0-optimal	A* is 0-optimal

Table 1: Optimality Results of D&P

a subset[2] of nodes expanded by A.

(1) A family of algorithms $\mathbf{B}$ is said to be 1-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $A \in \mathbf{A}$ and every problem instance $I \in \mathbf{I}$, there is an algorithm $B \in \mathbf{B}$ that expands a subset of nodes expanded by A.

(2) A family of algorithms $\mathbf{B}$ is said to be 2-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $B \in \mathbf{B}$, algorithm $A \in \mathbf{A}$ and problem instance $I \in \mathbf{I}$, if A does not expand a node expanded by B, then A necessarily expands a node that B does not expand.

(3) A family of algorithms $\mathbf{B}$ is said to be 3-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $A \in \mathbf{A}$ and every problem instance $I_1 \in \mathbf{I}$, if A does not expand a node expanded by some algorithm $B \in \mathbf{B}$, then, there exists some other instance $I_2 \in \mathbf{I}$, for which A necessarily expands a node that B does not expand.

We now summarize these definitions in quantifier notation. Let $exp(A, I)$ denote the set of nodes expanded by the algorithm A when solving the instance I. A family of algorithms $\mathbf{B}$ is said to be x-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ iff:

$\mathbf{x = 0}$. $\forall_{A\in\mathbf{A}, I\in\mathbf{I}}\ \forall_{B\in\mathbf{B}}\ exp(B, I) \subseteq exp(A, I)$

$\mathbf{x = 1}$. $\forall_{A\in\mathbf{A}, I\in\mathbf{I}}\ \exists_{B\in\mathbf{B}}\ exp(B, I) \subseteq exp(A, I)$[3]

$\mathbf{x = 2}$. $\forall_{A\in\mathbf{A}, I\in\mathbf{I}}\ \forall_{B\in\mathbf{B}}\ exp(A, I) \not\subset exp(B, I)$

$\mathbf{x = 3}$. $\forall_{A\in\mathbf{A}, I_1\in\mathbf{I}}\ \forall_{n_1\in exp(B,I_1)\setminus exp(A,I_1)}$
$\exists_{I_2\in\mathbf{I}}\ \exists n_2 \in exp(A, I_2) \setminus exp(B, I_2)$

D&P state that these types of optimality form a "hierarchy of decreasing strength". Therefore, once they state that, under certain assumptions, a 1-optimal algorithm does not exist, they do not need to state that a 0-optimal algorithm does not exist under the same assumptions.[4]

Optimality Results of D&P

Table 1 is a re-production of Figure 9 from page 531 of D&P. This table summarizes the optimality results proven by D&P (Section 4.3, pages 522-531). Consider, for example, the bottom left cell of this table. This cell states that A* possesses 0-optimality over the class of admissible algorithms and the class of non-pathological instances with a consistent heuristic. This means that, for such instances, every tie-breaking rule of A* expands a subset of nodes expanded by any admissible algorithm.

It is important to note that each result in Table 1 (except for the top left-most cell) endows A* with optimality in a certain context. For example, let us focus on the optimality of A* over the class of admissible algorithms (the left-most column of results). We see that:

(1) If we are interested in the optimality of A* without regard to the tie-breaking rule (0-optimality), then we have to restrict ourselves to the context of consistent heuristics and non-pathological instances.

(2) If we are interested in the optimality of A* up to tie-breaking (1-optimality), then we have to restrict ourselves to the context of consistent heuristics.

(3) Given an inconsistent heuristic, we cannot guarantee that A* will expand fewer (unique) nodes than other admissible algorithms. Rather, we have only a weaker claim that other algorithms cannot expand a proper subset of nodes expanded by A* (2-optimality).

In addition to this, although in practice we would like to have a guarantee that A* expands *fewer* nodes than other algorithms, none of the optimality results of D&P is stated in terms of the *number* of expanded nodes. Rather, these results are stated in terms of set relations.[5] Clearly, the claim "A* expands a subset of nodes expanded by algorithm B" is stronger than the claim "A* expands fewer nodes than algorithm B".

A** (the top left entry of Table 1) is an algorithm introduced by D&P. This algorithm computes $f(n)$ by maximizing over $(g + h)$ of all nodes on the current path from the

[2]Here the notion of subset is not strict, i.e. two identical sets are considered to be subsets of each other.

[3]The formulation of 1-optimality in D&P allows for a different interpretation: $\mathbf{x = 1}$. $\forall_{I\in\mathbf{I}}\ \exists_{B\in\mathbf{B}}\ \forall_{A\in\mathbf{A}}\ exp(B, I) \subseteq exp(A, I)$. However, the argument on page 524 for 1-optimality of A* for $\mathbf{I_{CON}}$ fits the definition that we bring in the main text.

[4]We were not able to see how it follows from the above definitions that every 1-optimal algorithm is automatically 2-optimal.

[5]The reason for this choice of definition of optimality is explained on page 521 of D&P. In our understanding, their argument is as follows. Suppose that we want to prove optimality of a family of algorithms $\mathbf{B}$ over a class of algorithms $\mathbf{A}$ ($\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$. Suppose that, for some instance, an algorithm $B \in \mathbf{B}$ expands fewer nodes than some algorithm $A \in \mathbf{A}$, but there is a node n that B expands, while A does not expand. Then we can form another input graph, where there is a large subtree with very cheap edges below n. For this new instance, B would expand all nodes in this subtree of n and expand more nodes than A. Thus, as soon as we allow the algorithms from $\mathbf{B}$ to expand a node that another algorithm does not expand, we cannot make any claims about the optimality of $\mathbf{B}$ with respect to the *number* of expanded nodes.

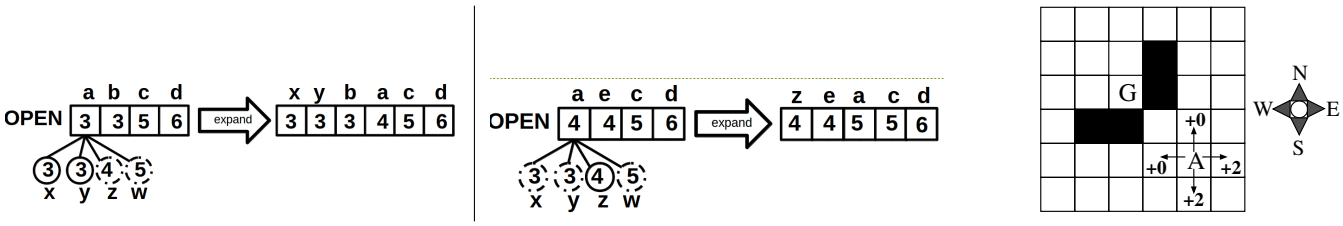

Figure 1: **Left and middle:** operation of PEA*, EPEA* and the new variants of EPEA*. **Right:** classification of operators.

start to n (inclusive). A more detailed discussion of this algorithm is beyond the scope of this paper.

EPEA*

In this section, we provide a concise description and a pseudo-code for EPEA* (Felner et al. 2012). EPEA* leverages the insight of its predecessor, PEA* (Yoshizumi, Miura, and Ishida 2000), that it is possible to never add surplus nodes to OPEN. EPEA* advances this idea and avoids even the generation of surplus nodes. EPEA* is best explained by describing PEA* first.

PEA* operates as follows. When expanding node n, PEA* first generates a list of *all* the children of n. However, only the children c with $f(c) = f(n)$ are added to OPEN. The remaining children are discarded and n is reinserted into OPEN with a new cost. This cost is computed as the smallest $f(c)$ that is greater than the current cost $f(n)$.

The authors of PEA* borrow the terminology first used in RBFS (Korf 1993). 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 the minimal static f-value among the children of n that were not added to OPEN is v.

The PEA* idea is shown in Figure 1 (left,middle). In Figure 1 (left), node a (with $F(a) = f(a) = 3$) is being expanded for the first time. All of its children (x, y, z, w) are generated. Then, its children with f-value of 3 (x and y) are inserted into OPEN. Node a is re-inserted into OPEN, but this time with a stored value $F(a) = 4$, the lowest cost among the remaining children (nodes w and z). When a is later chosen for expansion with $F(a) = 4$ as shown in Figure 1 (middle), all its children are generated. Those with $f < 4$ are discarded. Those with $f = 4$ are placed into OPEN (node z). If a has other children with $f > 4$, then a is re-inserted into OPEN with the lowest cost among them (5 in our example). Otherwise, a is closed.

Note that PEA* generates *all* children c of the node n being expanded, but puts into OPEN only the children with $f(c) = F(n)$. The main insight of EPEA* is that, for many domains, it is possible to generate *only* the children with $f(c) = F(n)$. EPEA* achieves this by classifying the operators applicable to any given node n based on the resulting change to the f-value: $\Delta f(c) = f(c) - f(n)$ and applying only the operators with $\Delta f(c) = F(n) - f(n)$ (which is equivalent to $f(c) = F(n)$). EPEA* uses a domain-dependent *Operator Selection Function* (OSF) which receives as input a node n and a value v. The OSF has two outputs: **(1)** a list of children c of n with $\Delta f(c) = F(n) - f(n)$ and **(2)** $F_{next}(n)$ — the smallest f-value among the children with $\Delta f(c) > F(n) - f(n)$.

Procedure 1 EPEA* and OGA*

```
1: Generate the start node n_s
2: For OGA*: if n_s is goal then exit                    // found optimal!
3: Compute h(n_s) and set F(n_s) ← f(n_s) ← h(n_s)
4: Put n_s into OPEN
5: while OPEN is not empty do
6:     For EPEA*:
7:         Get n with lowest F(n) from OPEN
8:         if n is goal then exit                        // found optimal!
9:         Set (N, F_next(n)) ← OSF(n, F(n)).
10:    For OGA*:
11:        Get n with lowest (F(n), bit(n)) from OPEN
12:        if F(n) = f(n) then
13:            Set f_g(n) ← g(n) + EGT(n)
14:            Set bit(n) ← 1
15:        end if
16:        if F(n) = f_g(n) then exit                    // found optimal!
17:        Set (N, F_next(n)) ← OSF(n, F(n)).
18:        if F_next(n) = f_g(n) then set bit(n) ← 0
19:    for all c ∈ N do
20:        Compute h(c), set g(c) ← g(n) + cost(n, c) and
           f(c) ← g(c) + h(c)
21:        Check for duplicates
22:        Set F(c) ← f(c) and put c into OPEN
23:    end for
24:    if F_next(n) = ∞ then
25:        Put n into CLOSED
26:    else
27:        Set F(n) ← F_next(n) and re-insert n into OPEN
28:    end if
29: end while
30: exit                                                 // There is no solution.
```

A simple example a classification of operators for the pathfinding domain on a 4-connected grid is shown in Figure 1 (right). Assume that all moves cost 1 and the heuristic function is *Manhattan Distance* (MD). Suppose that the goal node is located *northwest* of the current location. It is easy to see that f-value stays the same when going *north* or *west* and increases by 2 when going *south* or *east*. Similar classifications (though more complex) for the Pancake puzzle, the Rubik's Cube and the Multi-Agent Pathfinding domains are described in (Felner et al. 2012). For the rest of this paper, we will treat OSF as a black box. The reader who wants to understand in depth how OSF operates is referred to (Felner et al. 2012).

We now explain EPEA* with the pseudo-code in Procedure 1. Note that this pseudo-code shows both EPEA* and OGA*, which is one of the new variants of EPEA* introduced in this paper. In this section, we ignore the parts of the pseudo-code that are specific to OGA*.

When EPEA* generates a new node, it sets the new node's stored value to be equal to its static value (lines 3, 22). Nodes are expanded in the order of the best stored value (line 7). When expanding n, EPEA* calls OSF to generate only the children with $f(c) = F(n)$ (line 9).[6] These children are guaranteed to be useful. Also, OSF returns the next stored value of n, denoted by $F_{next}(n)$ (line 9). If n is later re-expanded, this value will be passed to OSF as the desired f-value of the children ($f(c)$). n is re-inserted into OPEN with this new stored value. If all children of n have been inserted into OPEN ($F_{next}(n) = \infty$), then n is put into CLOSED (line 25).

We demonstrate the operation of EPEA* with the example in Figure 1 (left and middle) where node a is being expanded. The first expansion of a (with $F(a) = f(a) = 3$) is shown in Figure 1 (left). EPEA* uses OSF to generate only the children with f-value of 3 (x and y) and inserts them into OPEN. OSF returns $F_{next}(a) = 4$, which is the lowest cost among the remaining children (nodes z and w). Node a is re-inserted into OPEN, but this time with the new stored value $F(a) = 4$.

Suppose that a is later chosen for expansion with $F(a) = 4$ as shown in Figure 1 (middle). EPEA* uses OSF to generate only the child z with f-value of 4 and inserts it into OPEN. If a has other children with $f > 4$, then lowest cost among them (5 in our example) would be returned by the OSF and a would be re-inserted into OPEN with $F(a) = 5$. If not, then OSF would return infinity and a would be closed.

Let us consider the first expansion of each node expanded by EPEA*. These are the same expansions that A* performs. Also, EPEA* performs these expansions in the same order as A*. Therefore, EPEA* is optimal with respect to the number of *unique* node expansions.

It is important to have in mind two points about EPEA* to put the theoretical results of this paper in proper context:

1. A* closes a node immediately after having generated its children. In contrast, EPEA* can expand a node many times before closing it. This may result in a much larger OPEN compared to A*. Furthermore, although EPEA* is optimal with respect to the number of *unique* node expansions, it may expand the same node many times. The variants of EPEA* introduced below have these limitation as well. Therefore, the theoretical optimality properties of the new variants do not necessarily guarantee that they will perform better (in terms of execution time) than A* in practice.
2. We will consider OSF to be a black box throughout the rest of the paper. This assumption is necessary to establish the theoretical results. However, the performance of OSF may have impact on the actual time performance of the new variants.

[6] If the heuristic is inconsistent, then, during the first expansion of n, another OSF is called, which generates all nodes with $f(c) \leq F(n)$. We assumed a consistent heuristic in order to not complicate the pseudo-code.

New Variants of EPEA*

In this section, we introduce two novel variants of EPEA*: *Optimal Generation A** (OGA*) and *Simple OGA** (SOGA*). OGA* and SOGA* are called *optimal*, because we will prove in the theoretical section below that these algorithms possess optimality with respect to the generated nodes. OGA* and SOGA* are called variants of EPEA* because they possess the main feature of EPEA* – using a domain-specific OSF to generate only the children c of the node n being expanded with $f(c) = F(n)$.

OGA*

Let us start by noting that EPEA* can be enhanced based on the following observation. When EPEA* generates the goal node g with $f(g) = C^*$, it must be that the node n being expanded has $F(n) = C^*$. Also, there are no nodes in OPEN that have a smaller stored value. Therefore, EPEA* does not have to put the goal into OPEN in order to later expand it. Rather, EPEA* can stop as soon the goal is generated.

In this section, we introduce a variant of EPEA* called *Optimal Generation A** (OGA*), which stops even before the goal node is generated – when the parent of the goal (denote this parent by p) with $F(p) = C^*$ is expanded, but before any of the children c of p with $F(n) = C^*$ (including the goal) are generated. However, the main feature of OGA* is a tie-breaking rule that guarantees that the parent of the goal is expanded as soon as possible.

Let us go through the parts of the pseudo-code in Procedure 1 that are specific to OGA*. For each node n, OGA* stores, in addition to $f(n)$ and $F(n)$, two more values:

(1) $f_g(n)$ is computed when n is expanded for the first time by the *enhanced goal test* (EGT) (line 13). EGT checks whether there is an edge in the state graph from n to the goal (how this is done is discussed in the next paragraph). If there is no such edge, EGT returns infinity. If there is such an edge, EGT returns the cost of the cheapest such edge. Thus, when $f_g(n) = g(n) + EGT(n)$ is finite, it is equal to the cost of reaching the goal when the goal is a neighbor of n. If the heuristic is inconsistent, then EGT has to be invoked each time n is re-expanded due to inconsistency (i.e. with a new g-value).

Clearly, generating all children of n in order to perform EGT would undermine the purpose of EPEA*. Therefore, just like OSF, EGT has to be implemented in a domain-dependent manner. There are two general approaches to implementing EGT. First, EGT can operate similarly to OSF – by scanning the operators applicable to n and looking for operators than decrease the heuristic value by $h(n)$ resulting in $h(c) = 0$. In fact EGT can be integrated with the OSF. Second, we can compute the list of the goal's neighbors before the search begins. With each such state, we would store the cost of the cheapest edge from that state to the goal. EGT would work by looking for n among these pre-computed states.

(2) $bit(n)$ is a bit showing whether $F(n)$ is equal to $f_g(n)$. This bit is set in lines 14 and 18.

Just like EPEA*, OGA* expands nodes in the order of the best stored value. It can use any tie-breaking rule with one restriction: nodes with $F(n) = f_g(n)$ are preferred (line 11). This guarantees that, if there are no nodes n in OPEN with $F(n) < C^*$, a node p with $F(p) = C^*$ from which there is a direct edge to the goal will be chosen for expansion if such a node p is in OPEN. Once such a node p is chosen for expansion, the algorithm can halt (line 16). In the theoretical section below, we will see that this guarantee endows OGA* with strong optimality properties with respect to the generated nodes.

Let us see how OGA* would operate for the example in Figure 1 (left and middle). Suppose that w is the goal node. When OGA* expands node a for the first time as shown in Figure 1 (left), it would use EGT to determine that a has a goal neighbor w with $f(w) = 5$ and set $f_g(a) = 5$. OGA* would also set $bit(a) = 1$. After expanding a for the second time as shown in Figure 1 (middle), OGA* would determine that $F_{next}(a) = 5$ is equal to $f_g(a)$ and set $bit(a) = 0$. Once no nodes with F-value 4 or less remain in OPEN, a will be the first node to be expanded and, according to line 16 of Procedure 1, OGA* will halt. This is because OGA* has established that reaching a through the current path leads in an optimal way to the goal node which is one of the neighbors of a.

SOGA*

We now introduce our second new variant of EPEA* – *Simple OGA** (SOGA*). SOGA* operates like EPEA* (i.e. the pseudo-code is identical to Procedure 1) with one exception: when SOGA* (re-)expands node n that has children c with $f(c) = F(n)$, it generates only one such child. If there are other children $f(c) = F(n)$, then their generation is postponed to the future re-expansions of n and n is re-inserted into OPEN with an unchanged F-value. SOGA* may use a tie-breaking rule to decide which child with $f(c) = F(n)$ to generate at each (re-)expansion of n.

Let us see how SOGA* would operate for the example in Figure 1 (left and middle). When SOGA* expands node a (with $F(a) = f(a) = 3$) for the first time as shown in Figure 1 (left), the OSF would generate only one of the children (x or y) with f-value of 3. Suppose that, according to the tie-breaking rule being used, x was generated. This OSF would also return $F_{next}(a) = 3$, which is the f-value of the next child with the required $f(c)$ (y in our example). Node a is re-inserted into OPEN with the unchanged stored value $F(a) = 3$. During the next expansion of a (Figure 1 (middle)), the OSF would generate the child y and return $F_{next}(a) = 4$, which is the lowest cost among the remaining children (nodes z and w). Node a is re-inserted into OPEN, but this time with the new stored value $F(a) = 4$.

In practice, SOGA* has a large run-time overhead compared to EPEA*: in order to generate b children of node n, SOGA* has to expand n exactly b times. However, we will see in the theoretical section below that, in contrast to EPEA*, SOGA* possesses optimality properties with respect to the number of generated nodes. This is because, once it expands a node n with $F(n) = C^*$, it may (that is, for some tie-breaking rule) avoid generating any node that is not on the optimal solution path.

Optimality of (S)OGA* With Respect to the Generated Nodes

We now consider the optimality of OGA* with respect to the generated nodes over the same classes of algorithms and instances, for which the optimality of A* with respect to the expanded nodes was established by D&P. By generating a node we mean forming a data structure corresponding to a search state. In the context of the following optimality results, two nodes corresponding to the same state are considered to be the same node, no matter how many times this node is generated by expanding the same or different nodes.

We will distinguish four types of optimality that are similar to the types of optimality used by D&P. Namely, we obtain the new types by substituting all occurrences of the word "expanded" by the word "generated". In order to distinguish the new types from the old ones, we append "prime" to their number. Also, we will write "prime optimality" to refer to optimality with respect to the generated nodes.

(0') A family of algorithms $\mathbf{B}$ (such as OGA* with all possible tie-breaking rules) is said to be 0'-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $A \in \mathbf{A}$ and every problem instance $I \in \mathbf{I}$, every algorithm $B \in \mathbf{B}$ generates a subset of nodes generated by A.

(1') A family of algorithms $\mathbf{B}$ is said to be 1'-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $A \in \mathbf{A}$ and every problem instance $I \in \mathbf{I}$, there is an algorithm $B \in \mathbf{B}$ that generates a subset of nodes generated by A.

(2') A family of algorithms $\mathbf{B}$ is said to be 2'-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $B \in \mathbf{B}$, algorithm $A \in \mathbf{A}$ and problem instance $I \in \mathbf{I}$, if A does not generate a node generated by B, then A necessarily generates a node that B does not generate.

(3') A family of algorithms $\mathbf{B}$ is said to be 3'-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ if, for every algorithm $A \in \mathbf{A}$ and every problem instance $I_1 \in \mathbf{I}$, if A does not generate a node generated by some algorithm $B \in \mathbf{B}$, then, there exists some other instance $I_2 \in \mathbf{I}$, for which A necessarily generates a node that B does not generate.

We now summarize these definitions in quantifier notation. Let $gen(A, I)$ denote the set of nodes generated by the algorithm A when solving the instance I. A family of algorithms $\mathbf{B}$ is said to be x'-optimal over a class of algorithms $\mathbf{A}$ (such that $\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$ iff:

$\mathbf{x = 0}$. $\forall_{A \in \mathbf{A}, I \in \mathbf{I}} \forall_{B \in \mathbf{B}}\ gen(B, I) \subseteq gen(A, I)$

$\mathbf{x = 1}$. $\forall_{A \in \mathbf{A}, I \in \mathbf{I}} \exists_{B \in \mathbf{B}}\ gen(B, I) \subseteq gen(A, I)$

$\mathbf{x = 2}$. $\forall_{A \in \mathbf{A}, I \in \mathbf{I}} \forall_{B \in \mathbf{B}}\ gen(A, I) \not\subset gen(B, I)$

$\mathbf{x = 3}$. $\forall_{A \in \mathbf{A}, I_1 \in \mathbf{I}} \forall_{n_1 \in gen(B, I_1) \setminus gen(A, I_1)}$
$\exists_{I_2 \in \mathbf{I}} \exists n_2 \in gen(A, I_2) \setminus gen(B, I_2)$

We now move to presenting our new results about the prime optimality of OGA*.

	$\mathbf{A_{AD}}$	$\mathbf{A_{gc}}$	$\mathbf{A_{bf}}$
$\mathbf{I_{AD}}$	**?**	SOGA* is 1'-optimal (L. 4)	SOGA* is 1'-optimal (L. 4)
	?	no 0'-optimal exists (L. 3) under VE assumption	no 0'-optimal exists (L. 3) under VE assumption
$\mathbf{I^{-}_{AD}}$	OGA* is 2'-optimal (L. 2) **?**	OGA* is 0'-optimal (L. 2)	OGA* is 0'-optimal (L. 2)
$\mathbf{I_{CON}}$	SOGA* is 1'-optimal (L. 4)	SOGA* is 1'-optimal (L. 4)	SOGA* is 1'-optimal (L. 4)
	no 0'-optimal exists (L. 3) under VE assumption	no 0'-optimal exists (L. 3) under VE assumption	no 0'-optimal exists (L. 3) under VE assumption
$\mathbf{I^{-}_{CON}}$	OGA* is 0'-optimal (L. 2)	OGA* is 0'-optimal (L. 2)	OGA* is 0'-optimal (L. 2)

Table 2: Optimality Results for OGA* and SOGA*.

The New Results

Our new optimality results are summarized in Table 2. For each entry in Table 2, we provide, in parentheses, the number of lemma (below) that proves the result. The bold question marks in the left-most column of Table 2 mean that establishing the corresponding optimality results remains an open research question.

Optimality of OGA* with respect to classes of non-pathological instances (rows 2 and 4 of Table 2) follows the following lemma:

Lemma 1. *For non-pathological instances, OGA* expands the same set of nodes that it generates.*[7]

Proof. Since every expanded node needs to be generated, we only need to show that, for non-pathological instances, OGA* expands all nodes that it generates.

Suppose that OGA* expanded node n and generated its child c. We consider three cases:

Case 1: $F(n) < C^*$. Since OGA* generates a child node c of a node n only if $f(c) = F(n)$, it must be that $f(c) < C^*$. Then c will surely be expanded before the search is completed by expanding a node with $F = C^*$ and $f_g = C^*$.

Case 2. $F(n) = C^*$ and there is a node n' in OPEN with $F(n') = f_g(n')$. This case is impossible. Namely, since every tie-breaking rule gives preference to nodes with $F(n) = f_g(n)$, it must be that $n = n'$. However, in such a case, the search would halt (line 16 of Procedure 1) and c would not be generated.

Case 3. $F(n) = C^*$ and there is no node n' in OPEN with $F(n') = f_g(n')$. We will show that this case is impossible either. Note that, for every optimal path, there is always a node in OPEN of OGA* that is on that path (this is easily verified by an inductive argument starting from the start node and following the expansion cycles). Since the instance is non-pathological, let P be an optimal path such that, for all non-goal nodes m on P, we have $f(m) < C^*$. Let v be the most advanced (i.e. the greatest g-value) node in OPEN that is on P. Note that v has a neighbor u, which is the not yet generated next node on P. If u is the goal, then, contrary to our assumption, we would have $f_g(v) = C^*$. Therefore, u is not a goal, which means that $f(u) < C^*$, which implies $F(v) < C^*$. However, this contradicts our assumption that n with $F(n) = C^*$ was chosen for expansion. This completes the proof. □

[7]We ignore the trivial instances with the start node being the goal, for which the start node is generated, but not expanded.

The following statement follows directly from Lemma 1 and corresponds to the results in rows 2 and 4 of Table 2:

Lemma 2. *If A* is x-optimal (where x can be 0,1 or 2) over a class of algorithms* **A** *and a class of non-pathological instances* **I***, then OGA* is x'-optimal over these classes.*

Proof. Suppose that A* is x-optimal (where x can be 0,1 or 2) over a class of algorithms **A** and a class of non-pathological instances **I**. Suppose that, contrary to our claim, an algorithm $A \in \mathbf{A}$ is a counter-example to x'-optimality of OGA* over these classes. We consider the three possible values for x in turn.

Case $x = 0$**.** Then, for some non-pathological problem instance, there is a node n that A does not generate, but some tie-breaking rule of OGA* generates. By Lemma 1, OGA* expands n. Also, since every expanded node has to be generated first, A does not expand n. Therefore, A is a counter-example to 0-optimality of OGA*. Since OGA* expands the same unique nodes as A*, A is a counter-example to 0-optimality of A*. We reached a contradiction.

Case $x = 1$**.** Then, for some non-pathological problem instance, for each tie-breaking rule of OGA*, there is a node n that OGA* generates, but A does not generate. Again, this means that OGA* (and, by implication, A*) expands n, while A does not expand n. Therefore, A is a counter-example to 1-optimality of A*. We reached a contradiction.

Case $x = 2$**.** Then, for some non-pathological problem instance and some tie-breaking rule of OGA*, A generates a strict subset of nodes generated by OGA*. Since **(1)** A expands a subset of nodes that it generates and **(2)** OGA* expands the same nodes that it generates, we have that A expands a strict subset of nodes expanded by OGA* (and, by implication, A*). Therefore, A is a counter-example to 2-optimality of A*. We reached a contradiction. □

Recall that D&P proved that their optimality results are tight. That is, if they proved that A* is 1-optimal over a certain class of algorithms and instances, they also proved that no 0-optimal algorithm exists. We would like to prove a similar result for prime optimality. We will do this by proving that, whenever an 0-optimal algorithm does not exist, there does not exist an 0'-optimal algorithm either. This result corresponds to the second line in the first and third rows of Table 2.

We were able to prove this result only under the following assumption that limits the classes of algorithms under consideration. We call it the *Valid Expansion (VE)* assumption:

1. An algorithm will not choose a node n for expansion if n clearly does not have children that meet the generation criteria. For example, if an algorithm has a rule not to generate nodes at depth $\geq d$ in the search tree, then such an algorithm will not expand a node at depth $d-1$.
2. Let n be the node chosen for expansion. An algorithm will not use a generation criteria that is based on:
   (a) State identity of the children of n.
   (b) Branching factor of n. An example of such a criteria would be the rule not to generate any children of n if the branching factor of n is greater than a certain threshold.
   (c) Incoming arcs to the children of n.

Lemma 3. *Whenever no 0-optimal algorithm that satisfies the VE assumption over a class of algorithms* $\mathbf{A}$ *and a class of instances* $\mathbf{I}$ *exists, there does not exist an 0'-optimal algorithm that satisfies the VE assumption over these classes.*

Proof. It will suffice to prove that, whenever a family $\mathbf{B}$ of algorithms that satisfies the VE assumption is $0'$-optimal over a class of algorithms $\mathbf{A}$ ($\mathbf{B} \subseteq \mathbf{A}$) and a class of instances $\mathbf{I}$, the family $\mathbf{B}$ is also 0-optimal over these classes. Suppose that, contrary to our claim, an algorithm $A \in \mathbf{A}$ is a counter-example to 0-optimality of $\mathbf{B}$ over these classes.

Then, for some problem instance $I \in \mathbf{I}$, A does not expand a node n that some algorithm $B \in \mathbf{B}$ expands. We shall modify I to construct an instance I' with the following properties:

1. I' belongs to the same class of instances as I. That is **(a)** if I is non-pathological, then I' is non-pathological and **(b)** if h is consistent in I, then it remains consistent in I'.
2. n has a child c such that **(a)** B generates c when solving I' and **(b)** A does not generate c when solving I'.

Such a construction will contradict 0'-optimality of $\mathbf{B}$ and complete the proof.

We need to consider the following cases and construct I' accordingly for each case:

Case 1. There is a child c of n in I such that **(a)** B generates c when solving I and **(b)** A does not generate c when solving I. Then we take I' to be identical to I.

Case 2. There is a child c' of n in I such that **(a)** B generates c' when solving I and **(b)** A generates c' when solving I as well (this is possible if there is a path from the start state to c' that does not pass through n). We construct I' by appending I with a child c of n, such that $cost(n,c) = cost(n,c')$ and $h(c') = h(c)$. Furthermore, c has outgoing arcs to the same states to which c' has outgoing arcs. We complete the construction of I' by deleting the arc from n to c' and putting c into the same position in the adjacency list of n where c' used to be. By the VE assumption, B will generate c when solving I'. Also, since A does not expand n, it cannot generate c.

Case 3. B does not generate any children of n when solving I. We construct I' by appending I with a child c of n, such that $cost(n,c) = 0$ and $h(c) = h(n)$. Furthermore, c has outgoing arcs to the same states to which n has outgoing arcs. Thus, the only possible distinction between c and n is the depth in the search tree. By the VE assumption, B will generate c when solving I'. Also, since A does not expand n, it cannot generate c.

In all three cases, it is easy to see that I' belongs to the same class of instances as I. □

Finally, we prove that, for pathological instances, SOGA* is 1'-optimal over the classes of algorithms for which 1-optimality of A* was proven by D&P (first and third rows of Table 2).

Lemma 4. *Whenever A* is 1-optimal over a class of algorithms* $\mathbf{A}$ *and a class of instances* $\mathbf{I}$*, SOGA* is 1'-optimal over these classes.*

Proof. Suppose that A* is 1-optimal over a class of algorithms $\mathbf{A}$ and a class of instances $\mathbf{I}$. Then, for every algorithm $A \in \mathbf{A}$ and every problem instance $I \in \mathbf{I}$, there is a tie-breaking rule T of A* that expands a subset of nodes expanded by A. Our claim follows from the fact that we can choose an appropriate tie-breaking rule for generations of SOGA*, such that SOGA* will generate only and all the nodes that will be expanded by the tie-breaking rule T. Then, for the same instance I, this tie-breaking rule for generations and T applied together will result in SOGA* generating the subset of nodes generated by A. □

Finally, let us provide a simple example that shows why none of the optimality results proven for OGA* and SOGA* holds for EPEA*. Suppose an instance where the goal g has only one neighbor p. Suppose that, besides the goal, p also has neighbors n_1 and n_2. Lastly, suppose that $f(p) = 9$, $f(g) = f(n_1) = f(n_2) = 10$ and that p was expanded and received $F(p) = 10$. Note that EPEA* will not generate the goal g until no nodes n with $F(n) \leq 9$ in OPEN remain and p gets expanded. At this point, EPEA* will generate all neighbors c of p with $f(c) = 10$, namely g, n_1 and n_2. However, since every tie-breaking rule of EPEA* gives preference to the goal, only g will get expanded, while n_1 and n_2 are only generated. On the other hand, we know that any tie-breaking rule of OGA* generates neither n_1 nor n_2 (actually, it does not generate g either unless g is the start node) and that SOGA* may avoid generating n_1 or n_2 or both.

Conclusions

We have leveraged the results about the optimality of A* with respect to the expanded nodes by (Dechter and Pearl 1985) (D&P) to show that two new variants of EPEA*, OGA* and SOGA*, are optimal with respect to the generated nodes. The notions of optimality are very similar to the notions that D&P use to formalize the optimality of A*.

Two open theoretical questions remain, which correspond to the question marks in Table 2:

(1) Is there an algorithm that is 1'-optimal over the class of admissible algorithms and the class of non-pathological instances with an admissible heuristic?

(2) Are there algorithms the would provide a counter-example to Lemma 3 when the VE assumption is removed?

(3) Can any prime optimality result be established over the class of admissible algorithms and the class of pathological instances with an admissible heuristic?

Finally, in this paper, we discussed only the theoretical properties of OGA* and simple OGA*. It remains to see whether practical benefits (i.e. reduction in running time) can be gained by solving difficult instances of domains with a large branching factor with these new variants of EPEA*.

Acknowledgements

This research was supported by the Israeli Science Foundation (ISF) grant 305/09 to Ariel Felner and the Natural Sciences and Engineering Research Council of Canada grant to Jonathan Schaeffer.

References

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery* 32(3):505–536.

Felner, A.; Goldenberg, M.; Sharon, G.; Stutervant, N.; Stern, R.; Beja, T.; Schaeffer, J.; and Holte, R. 2012. Partial-expansion A* with selective node generation. In *Proceedings of AAAI*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible treesearch. *Artificial Intelligence* 27(1):97–109.

Korf, R. E. 1993. Linear-space best-first search. *Artificial Intelligence* 62(1):41–78.

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