Journal of Artificial Intelligence Research 67 (2020) 81-113 Submitted 03/2019; published 01/2020

The 2^k Neighborhoods for Grid Path Planning

Nicolás Rivera NICOLAS.RIVERA@CL.CAM.AC.UK
Computer Laboratory,
University of Cambridge,
Cambridge, UK

Carlos Hernández CARLOS.HERNANDEZ.U@UNAB.CL
Nicolás Hormazábal NIC.HORMAZABAL@UANDRESBELLO.EDU
Departamento de Ciencias de la Ingeniería,
Universidad Andrés Bello,
Santiago, Chile

Jorge A. Baier JABAIER@ING.PUC.CL
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Grid path planning is an important problem in AI. Its understanding has been key for the development of autonomous navigation systems. An interesting and rather surprising fact about the vast literature on this problem is that only a few neighborhoods have been used when evaluating these algorithms. Indeed, only the 4- and 8-neighborhoods are usually considered, and rarely the 16-neighborhood. This paper describes three contributions that enable the construction of effective grid path planners for extended 2^k-neighborhoods; that is, neighborhoods that admit 2^k neighbors per state, where k is a parameter. First, we provide a simple recursive definition of the 2^k-neighborhood in terms of the 2^{k-1}-neighborhood. Second, we derive distance functions, for any $k \geq 2$, which allow us to propose admissible heuristics that are perfect for obstacle-free grids, which generalize the well-known Manhattan and Octile distances. Third, we define the notion of canonical path for the 2^k-neighborhood; this allows us to incorporate our neighborhoods into two versions of A*, namely Canonical A* and Jump Point Search (JPS), whose performance, we show, scales well when increasing k. Our empirical evaluation shows that, when increasing k, the cost of the solution found improves substantially. Used with the 2^k-neighborhood, Canonical A* and JPS, in many configurations, are also superior to the any-angle path planner Theta* both in terms of solution quality and runtime. Our planner is competitive with one implementation of the any-angle path planner, ANYA in some configurations. Our main practical conclusion is that standard, well-understood grid path planning technology may provide an effective approach to any-angle grid path planning.

1. Introduction

Grid path planning is one of the most well-known problems in AI. It arises naturally when modeling the problem of goal-directed navigation over a two-dimensional space as a graph search problem over a grid in which each cell of the grid can either be an obstacle or be free.

Grid path planning has a number of applications, ranging from robotics (Lee & Yu, 2009) and video games (Björnsson, Enzenberger, Holte, & Schaeffer, 2005). Furthermore, it still captures significant attention from the AI community. Notably, three editions of the Grid Path Planning Competition (GPPC) have recently been held (Sturtevant et al., 2015), putting to test the latest advances in the area (Botea & Harabor, 2013; Harabor & Grastien, 2011; Uras, Koenig, & Hernández, 2013).

Research on grid path planning has focused on simple 4-neighbor grids in which cardinal moves are allowed, and 8-neighbor grids that extend 4-neighbor moves with diagonal moves. Perhaps the main reason why this is so is that these neighborhoods are simple to implement and that good heuristics are known for them. Indeed, the Manhattan and Octile distances (Sturtevant & Buro, 2005) are perfect heuristics for, respectively, 4- and 8-neighbor obstacle-free grids. Although distance functions for the 16-neighborhoods have been discovered and studied by researchers of the Computer Vision community (Marchand-Maillet & Sharaiha, 1997), to our knowledge, evaluations of grid path planning over 16-neighborhoods (e.g. Nash, 2012; Aine & Likhachev, 2016) have never considered these heuristics, using the Euclidean distance (ED) instead.

Perfect heuristics for obstacle-free grids allow planners like A* to find solutions more quickly. They frequently are a key enabler of other techniques for grid path planning. One example is the approach by Uras et al. (2013), winner of the 2013 GPPC, optimal track, which relies heavily on the Octile distance to compute subgoal graphs that are later exploited for fast path planning. Another example is FRIT (Rivera, Illanes, Baier, & Hernández, 2014), a state-of-the-art real-time heuristic search path planning algorithm whose performance relies on the construction of a so-called *ideal tree* for which the Octile/Manhattan heuristic is used.

An important issue when finding paths with 4- and 8-neighbor grids is suboptimality with respect to an any-angle path. This issue has also been referred to as *digitization bias* (Tsitsiklis, 1995; Hew, 2017). Intuitively, because only a few moves are allowed to generate successors, paths cannot bend with the angles needed to achieve optimality. Bailey, Tovey, Uras, Koenig, and Nash (2011), indeed, studied problem of suboptimality of 4- and 8-neighbor path planning in detail, and established that the cost of 4- and 8-neighbor optimal solutions can be, at most, a factor of $\sqrt{2} \approx 1.41$ and $\sqrt{4 - 4\sqrt{2}} \approx 1.08$ away from the cost of any-angle optimal solutions, respectively.[1]

To find solutions of better quality while still applying graph search algorithms, researchers have extended A* with the ability of finding moves with a greater angle diversity (e.g., Daniel, Nash, Koenig, & Felner, 2010), and proposed algorithms that can directly consider paths that can utilize arbitrary angles (e.g, Harabor, Grastien, Öz, & Aksakalli, 2016). Another option, which has not been considered in depth in the literature, is increasing the size of the neighborhood.

In this paper we study grid path planning on the 2^k-neighborhoods, which are neighborhoods which, given a parameter k, define 2^k moves. Our first contribution is a definition of such neighborhoods. A notable property we prove is that the radius of the smallest square that contains the moves is given by the Fibonacci series. Furthermore, we derive and prove the correctness of an algorithm that returns the distance between any two points

1. These results are obtained by Bailey et al. (2011) when the vertices of the search graph are placed at the corners of the grid cells.

of an obstacle-free grid over the 2^k-neighborhood. Our proof and construction work for any k, thus generalizing the Manhattan and Octile distances, and the distance function of Marchand-Maillet and Sharaiha. Our proof, however, seems much shorter and simpler than the proof by Marchand-Maillet and Sharaiha for the 16-neighborhood. Finally, we define canonical orderings (Harabor & Grastien, 2011; Sturtevant & Rabin, 2016) for the 2^k-neighborhood. Canonical ordering is a technique at the core of Jump Point Search (JPS) (Harabor & Grastien, 2011), which is among the fastest search algorithms for 8-connected grid path planning.

We implemented three 2^k path planners: regular A*, A* with our canonical orderings, and a 2^k version of JPS. We evaluate them over standard benchmarks. We test different values of k, and compared our 2^k-tile heuristic with the ED. The runtime of Canonical A* and JPS scales with k while this does not hold for regular A*. Our heuristics yield faster search compared to the ED, for every neighborhood with regular A*, and up to the 8-neighborhood with Canonical A*. We compared also with the any-angle path planners Theta* (Daniel et al., 2010) and two implementations of ANYA (Harabor & Grastien, 2013; Harabor et al., 2016): a version by Uras and Koenig (2015) and the version reported by Harabor et al. (2016). The optimized version of ANYA (Harabor et al., 2016) outperforms our implementations in runtime, while we outperform Theta* in many of our configurations.

The research we report in this paper includes and extends a previously published AAAI-17 paper (Rivera, Hernández, & Baier, 2017). The following items describe material not included in the AAAI publication.

- We propose a new iterative heuristic function and prove its correctness. This function had been published before in another conference publication (Hormazábal, Díaz, Hernández, & Baier, 2017).
- We discuss and evaluate a 2^k version of JPS.
- We consider the path planning problem when the agent stands on the corners of the cells rather than in the middle. We adopt this different view because in 4- and 8-connected grids, this yields lower cost (optimal) solutions (Bailey et al., 2011).
- We extend the experimental results by showing the performance over different types of benchmarks (game maps, rooms, and random).
- We extend the theoretical results with an additional theorem (Theorem 10) on the consistency of our proposed heuristic, h_{2^k}. Furthermore, we provide complete proofs for all theorems.
- We compare against the any-angle path planners Theta* and ANYA over all benchmarks.

Next, we introduce background information and then define the 2^k-neighborhood. Then we derive our 2^k-tile heuristic. We continue proposing our canonical orderings, and finish with our empirical evaluation and conclusions.

2. Background

In this section we review the basics of grid path planning and introduce the most commonly used neighborhoods.

2.1 Grid Path Planning

An $N \times M$ *grid* is a tuple (C, O), where $C = \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid 0 \leq i < N, 0 \leq j < M\}$ is a set of *cells* and $O \subseteq C$ is the set of *obstacle cells*. The four *corners* of cell (i, j) are (i, j) (the lower-left corner), $(i + 1, j)$ (the lower-right corner), $(i, j + 1)$ (the upper-left corner), $(i + 1, j + 1)$ (the upper-right corner).

In the grid path planning literature, two assumptions have been made regarding where the position the agent may be located at while moving through the grid. The first, more traditional assumption, is that the agent is located in the center of the cell. In the second, the agent is located at one of the corners of some cell. In the rest of the paper, we adopt this second view, because (1) it simplifies the presentation when considering extended neighborhoods, and (2) paths found with using this assumption are shorter and closer to truly optimal paths (Bailey et al., 2011) in 4- and 8-connected grids. Another important reason to take this view is that Kramm, Rivera, Hernández, and Baier (2018) recently showed that when k increases and vertices are located at the corners of cells, the cost of optimal paths provably get closer to the any-angle optimal path, while this is not the case when vertices are located at the centers of cells. Of course, a consequence of this view is that the agent moves very close to obstacle cells; indeed, it 'touches' obstacles. While this may be seen as a drawback, it is consistent with the traditional any-angle literature (cf. Figure 2; Harabor et al., 2016).

The *search graph* associated to a grid (C, O) has one vertex for each of the corners of a cell in C. The edges of the search graph are determined by which moves can be performed by the agent. Each move is an ordered pair of integers. The set of moves is called *neighborhood*.

In the rest of the paper, we use ordered pairs to denote both moves and points in 2D. We interpret these pairs as 2D vectors. As such we assume the following identities hold: $(x, y) + (x'', y') = (x + x', y + y')$, and $c(x, y) = (cx, cy)$. We denote ordered pairs using boldface. We use the notation $[a, b]$ to refer to the real interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$, and $]a, b[$ to refer to $\{x \in \mathbb{R} \mid a < x < b\}$. The *interior* of a cell $\mathbf{c}$ is the set of points $\{\mathbf{c} + \mu(0, 1) + \nu(1, 0) \mid \mu, \nu \in]0, 1[\}$. The *border* of a cell $\mathbf{c}$ are those points in $\{\mathbf{c} + \mu(0, 1) + \nu(1, 0) \mid \mu, \nu \in [0, 1]\}$ that are not in the interior of $\mathbf{c}$. Finally, we say that (x, y) is *contained* by two moves $\mathbf{m}$ and $\mathbf{m'}$ iff there exist two non-negative numbers α and β such that $(x, y) = \alpha\mathbf{m} + \beta\mathbf{m'}$.

The set of points *visited* by applying a move $\mathbf{m}$ in $\mathbf{p}$ is $\{\mathbf{p} + \lambda\mathbf{m} \mid \lambda \in]0, 1[\}$. A move $\mathbf{m}$ is *applicable* in a vertex $\mathbf{u}$ iff:

1. $\mathbf{u} + \mathbf{m}$ is a vertex of the graph, and
2. points visited by $\mathbf{m}$ when applied in $\mathbf{u}$ do not contain:
   (a) points in the interior of an obstacle cell
   (b) points in the intersection of the borders of two cells[2]

2. Note that the intersection of the borders of two cells can be nonempty only if those cells are adjacent.

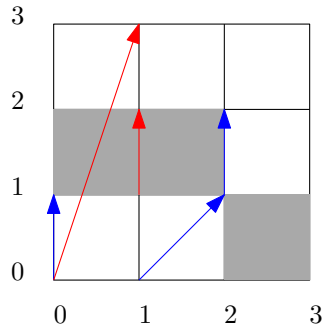


Figure 1: A 3×3 grid showing applicable moves (in blue) and inapplicable moves (in red). Move $(1, 3)$ is not applicable in vertex $(0, 0)$ because it visits points in the interior of cell $(0, 1)$. Moreover, move $(0, 1)$ is not applicable in vertex $(1, 1)$ because it visits points in the intersection of the borders of cells $(0, 1)$ and $(1, 1)$.

See Figure 1 for an illustration of moves that are applicable and not applicable.

The set of successors of a vertex $\mathbf{u}$ is defined as:

$$Succ_{\mathcal{N}}(\mathbf{u}) = \{\mathbf{m} + \mathbf{u} \mid \mathbf{m} \in \mathcal{N} \text{ and } \mathbf{m} \text{ is applicable in } \mathbf{u}\}.$$

A *path* from $\mathbf{u}$ to $\mathbf{v}$ over $\mathcal{N}$ is a sequence of vertices $\mathbf{v}_1\mathbf{v}_2 \cdots \mathbf{v}_n$ such that $\mathbf{v}_1 = \mathbf{u}$, $\mathbf{v}_n = \mathbf{v}$, and for every $i \in \{1, \ldots, n-1\}$ it holds that $\mathbf{v}_{i+1} \in Succ_{\mathcal{N}}(\mathbf{v}_i)$. Note that this definition implies that paths can squeeze through diagonally placed obstacles; in Figure 1, path $(1, 0)(2, 1)(2, 2)$ is legal. We say that a path $\mathbf{v}_1\mathbf{v}_2 \cdots \mathbf{v}_n$ is *generated* by the sequence of moves $\mathbf{m}_1\mathbf{m}_2 \cdots \mathbf{m}_{n-1}$ iff $\mathbf{v}_i + \mathbf{m}_i = \mathbf{v}_{i+1}$, for every $i \in \{1, \ldots, n-1\}$. Two vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ are *reachable over* $\mathcal{N}$ if there exists a path over $\mathcal{N}$ that starts with $\mathbf{v}_1$ and ends with $\mathbf{v}_2$.

The cost of a path $\sigma = \mathbf{v}_1 \cdots \mathbf{v}_n$ is $c(\sigma) = \sum_{i=1}^{n-1} \|\mathbf{v}_{i+1} - \mathbf{v}_i\|$. A path σ from $\mathbf{u}$ to $\mathbf{v}$ over $\mathcal{N}$ is *optimal* if for every path σ' from $\mathbf{u}$ to $\mathbf{v}$ over $\mathcal{N}$ it holds that $c(\sigma) \leq c(\sigma')$.

A grid path planning problem is a tuple $P = (C, O, \mathcal{N}, \mathbf{u}_{start}, \mathbf{u}_{goal})$, where (C, O) is a grid, $\mathcal{N}$ is a neighborhood, $\mathbf{u}_{start} \in G$ and $\mathbf{u}_{goal} \in G$ are, respectively, the start and goal vertices of the search graph associated with (C, O) for neighborhood $\mathcal{N}$. A *solution* (resp. *optimal solution*) for P is a path (resp. optimal path) over $\mathcal{N}$ from $\mathbf{u}_{start}$ to $\mathbf{u}_{goal}$ containing only moves in $\mathcal{N}$.

A concept that is useful below is that of a *convex corner of an obstacle*. A pair (x, y) is a convex corner of an obstacle if either:

1. (x, y) is the lower-left or upper-right corner of some obstacle cell and moves $(1, -1)$ and $(-1, 1)$ are applicable in (x, y), or

2. (x, y) is the upper-left or lower-right corner of some obstacle cell and moves $(1, 1)$ and $(-1, -1)$ are applicable in (x, y).

The definition above does not assume that the diagonal moves, such as $(1, -1)$, are part of the neighborhood. Thus this definition also applies to a 4-connected grid.

2.2 The 4- and 8-Connected Neighborhoods and Their Heuristics

The 4- and 8-connected neighborhoods have been used traditionally to evaluate grid path planning algorithms. The 4-connected neighborhood is defined as:

$$\mathcal{N}_4 = \{(i,j) \mid i,j \in \{-1,0,1\}, |i| + |j| = 1\},$$

effectively only allowing vertical and horizontal moves. The 8-connected neighborhood, in addition, extends the 4-connected neighborhoods with diagonal moves. It is defined as:

$$\mathcal{N}_8 = \{(i,j) \mid i,j \in \{-1,0,1\}, |i| + |j| > 0\}.$$

The *Manhattan distance* between two grid cells $\mathbf{u}$ and $\mathbf{v}$ is the cost of an optimal path over $\mathcal{N}_4$ between $\mathbf{u}$ and $\mathbf{v}'$, assuming the grid is obstacle-free. It is defined as $|\Delta x| + |\Delta y|$, where Δx and Δy satisfy $(\Delta x, \Delta y) = \mathbf{v} - \mathbf{u}$. Likewise, the *Octile distance*, given two pairs of cells $\mathbf{u}$ and $\mathbf{v}$, returns the cost of an optimal path over $\mathcal{N}_8$ assuming the set of obstacles is empty. It is defined as

$$\max\{|\Delta x|, |\Delta y|\} + (\sqrt{2} - 1)\min\{|\Delta x|, |\Delta y|\},$$

where Δx and Δy are defined as above. We denote the Manhattan and Octile distance between $\mathbf{u}$ and $\mathbf{v}$, respectively, as $h_4(\mathbf{u}, \mathbf{v})$ and $h_8(\mathbf{u}, \mathbf{v})$.

2.3 The Any-Angle Neighborhood

Any-Angle grid path planning allows the agent to move to any point in the grid that is line-of-sight. Therefore the any-angle neighborhood contains a move for reaching every vertex in the search graph.

3. The 2^k Neighborhoods

Below, we define the 2^k-neighborhoods such that they generalize the 4-, 8-, and 16-connected neighborhoods traditionally used in the grid path planning community. We define $\mathcal{N}_{2^k}$, for every $k \geq 2$, in terms of a series $\mathcal{Q}_0, \mathcal{Q}_1, \ldots$, which is such that $\mathcal{Q}_i$ is a sequence that contains the first-quadrant moves of $\mathcal{N}_{2^{i+2}}$. Its first element, $\mathcal{Q}_0$, contains the 4-neighborhood moves on the first (i.e., positive) quadrant; thus, $\mathcal{Q}_0 = \langle (1,0), (0,1) \rangle$. Now if $\mathcal{Q}_i = \langle \mathbf{a}_0, \ldots, \mathbf{a}_n \rangle$, $\mathcal{Q}_{i+1}$ is constructed from $\mathcal{Q}_i$ by inserting between each consecutive elements $\mathbf{a}_j$ and $\mathbf{a}_{j+1}$ the sum $\mathbf{a}_j + \mathbf{a}_{j+1}$. Formally, $\mathcal{Q}_{i+1} = \langle \mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{2n} \rangle$, where

$$\mathbf{b}_{2j} = \mathbf{a}_j, \text{when } 0 \leq j \leq n, \text{ and} \tag{1}$$

$$\mathbf{b}_{2j+1} = \mathbf{a}_j + \mathbf{a}_{j+1}, \text{when } 0 \leq j < n. \tag{2}$$

The first three elements of the series are:

$$\mathcal{Q}_0 = \langle (1,0), (0,1) \rangle$$
$$\mathcal{Q}_1 = \langle (1,0), (1,1), (0,1) \rangle$$
$$\mathcal{Q}_2 = \langle (1,0), (2,1), (1,1), (1,2), (0,1) \rangle$$

$\mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_5$ are also illustrated as the first-quadrant elements of Figure 2.

Observe that elements in $\mathcal{Q}_i$ are pairwise linearly independent. This is an important property because it says that moves are unique in the sense that they cannot be simulated by repeating other moves in the neighborhood. We can prove this fact by induction. Indeed, $\mathcal{Q}_0$ is such that its two elements are linearly independent. Moreover, every new element added to $\mathcal{Q}_{i+1}$ that was not in $\mathcal{Q}_i$ is linearly independent from each element in $\mathcal{Q}_i$, because it is the sum of two (linearly independent) elements of $\mathcal{Q}_i$. We formalize this as:

Proposition 1 *If $\mathbf{u}, \mathbf{v} \in \mathcal{Q}_i$, and there exists a k such that $\mathbf{u} = k\mathbf{v}$, then $\mathbf{u} = \mathbf{v}$.*

To define a neighborhood in terms of the first-quadrant moves, we simply map every move to all four quadrants. As a result, for the orthogonal moves ($(0,1)$ and $(1,0)$), we generate two new moves ($(0,-1)$ and $(-1,0)$) while each non-orthogonal (x,y) generates 3 additional moves that correspond to changing the signs of x and y. In short, for every natural i, we define:

$$\mathcal{N}_{2^{i+2}} = \{(kx, k'y) \mid (x,y) \text{ is in } \mathcal{Q}_i \text{ and } k, k' \in \{-1, 1\}\} \tag{3}$$

Henceforth we may abuse notation by interpreting $\mathcal{N}_{2^k}$ as a sequence of moves $\langle \mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{|\mathcal{N}_{2^k}|-1} \rangle$. When we do so, we assume the first element of the sequence is $(0,1)$, and that $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ are such that no $\mathbf{u} \in \mathcal{N}_{2^k}$ different from $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ is contained by $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$, for every $i \in \{0, \ldots, |\mathcal{N}_{2^k}| - 2\}$; that is, moves in $\mathcal{N}_{2^k}$ are ordered clockwise. In addition, if $\mathbf{m} = \mathbf{v}_j$, for some j, then $\mathbf{m} + 1$ and $\mathbf{m} - 1$ denote, respectively the move immediately clockwise (that is, $\mathbf{v}_{(j+1) \mod 2^k}$) and immediately counter-clockwise (that is, $\mathbf{v}_{(j-1) \mod 2^k}$) from $\mathbf{m}$. Finally, if $\mathbf{m} = v_j$ for some odd j, we say that $\mathbf{m}$ is an odd move, and when j is even, we say $\mathbf{m}$ is an even move.

Proposition 2 *The cardinality of $\mathcal{N}_{2^k}$ is 2^k, for every $k \geq 2$.*

Proof: Observe that $|\mathcal{Q}_0| = 2$ and that $|\mathcal{Q}_{i+1}| = 2|\mathcal{Q}_i| - 1$, which is a recurrence equation whose solution is $|\mathcal{Q}_i| = 2^i + 1$. For every non-orthogonal move in $\mathcal{Q}_i$ (of which there are $|\mathcal{Q}_i| - 2$) there are four moves in $\mathcal{N}_{2^{2+i}}$. Additionally, $\mathcal{N}_{2^{2+i}}$ contains 4 orthogonal moves, yielding a total of $4(|\mathcal{Q}_i| - 2) + 4$ moves. This yields $|\mathcal{N}_{2^{2+i}}| = 2^{i+2}$, for every $i \geq 0$. ■

A property of our definition is that the size of the smallest square in which all moves in $\mathcal{Q}_i$ can be circumscribed—defined below as the radius of the neighborhood—grows exponentially (see Figure 2); more precisely, it grows with the Fibonacci numbers. Before establishing the result, let us formally define the radius of a neighborhood.

Definition 3 *The* radius *of a neighborhood $\mathcal{N}$ is defined as:*

$$\text{radius}(\mathcal{N}) = \max\{\max\{|p|, |q|\} \mid (p,q) \in \mathcal{N}\} \tag{4}$$

Now, we define the Fibonacci series using the Fib function:

$$\text{Fib}(i) = \begin{cases} 1 & \text{if } i = 0 \text{ or } i = 1 \\ \text{Fib}(i-1) + \text{Fib}(i-2) & \text{if } i > 1. \end{cases}$$

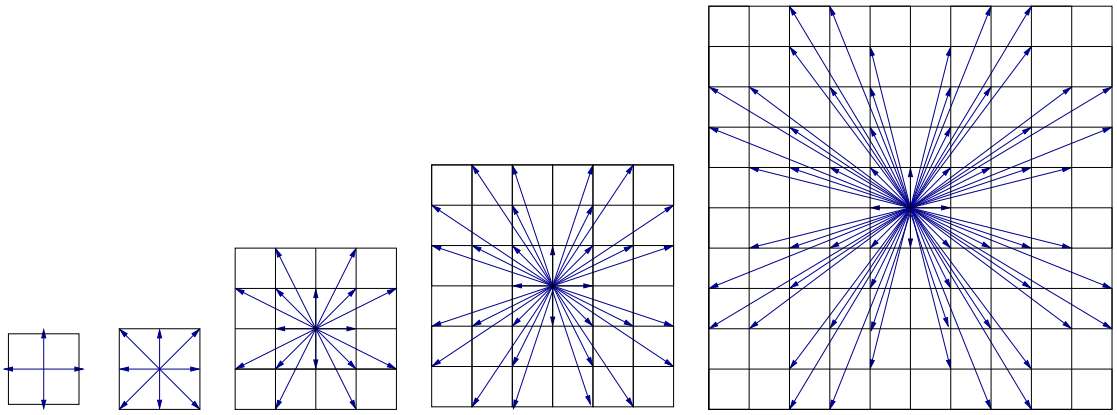


Figure 2: The 4-, 8-, 16-, 32-, and 64- neighborhoods.

Theorem 4 $\text{radius}(\mathcal{N}_{2^k}) = \text{Fib}(k-2)$.

To simplify notation, below we use $\mathcal{Q}^k$ to denote the element in position k of sequence $\mathcal{Q}$, where k may take values from 0 to $|\mathcal{Q}| - 1$. To prove Theorem 4, first observe that $\mathcal{Q}_i^k = \mathcal{Q}_{i-1}^{(k-1)/2} + \mathcal{Q}_{i-1}^{(k+1)/2}$ holds true, when k is an odd number. In addition, observe that $(k+1)/2$ and $(k-1)/2$ are consecutive and thus one of them is an *even* number. Therefore, if $(k+1)/2$ is even, then, via Equation (1):

$$\mathcal{Q}_i^k = \mathcal{Q}_{i-1}^{(k-1)/2} + \mathcal{Q}_{i-2}^{(k+1)/4}. \tag{5}$$

Otherwise, if $(k-1)/2$ is even:

$$\mathcal{Q}_i^k = \mathcal{Q}_{i-2}^{(k-1)/4} + \mathcal{Q}_{i-1}^{(k+1)/2}. \tag{6}$$

The proof of Theorem 4 is straightforward from the following two lemmata.

Lemma 5 *Each coordinate of any pair in $\mathcal{Q}_i$ is lower than or equal to* $\text{Fib}(i)$.

Proof: By induction on i, we verify by inspection that this is true in the base case ($i = 0$). Assume the property holds for every $i \in \{1, \ldots, n-1\}$. We slightly abuse notation below by saying $(x, y) \leq k$ when both $x \leq k$ and $y \leq k$ hold. Now, we prove that for every k, $\mathcal{Q}_n^k \leq \text{Fib}(n)$. We distinguish two cases: k is even, and k is odd. If k is even, then $\mathcal{Q}_n^k = \mathcal{Q}_{n-1}^{k/2}$. By the inductive hypothesis, $\mathcal{Q}_n^k \leq \text{Fib}(n-1)$ and therefore we conclude $\mathcal{Q}_n^k \leq \text{Fib}(n)$, because Fib is non-decreasing.

Now assume k is odd. Then we have two cases (Equations (5) and (6)) which are proven analogously. The proof follows in the same way for both cases so we simply assume:

$$\mathcal{Q}_n^k = \mathcal{Q}_{n-1}^{(k-1)/2} + \mathcal{Q}_{n-2}^{(k+1)/4} \tag{7}$$

By the induction hypothesis and the definition of Fib,

$$\mathcal{Q}_n^k \leq \text{Fib}(n-1) + \text{Fib}(n-2) = \text{Fib}(n),$$

which concludes the proof. ■

Lemma 6 *Let F_i be the series defined by $F_0 = 0$, and $F_i = 2F_{i-1} + (-1)^{i-1}$, for $i > 0$. Then the first coordinate of $\mathcal{Q}_i^{F_i}$ equals* Fib(i)*, for every i.*

Proof: The first observation is that F_i is formed by adding or subtracting 1 to an even number ($2F_{i-1}$) and thus is odd, for every $i > 0$. Now the proof proceeds by induction on i. Note that for the base case ($i = 0$) the property holds. Assume the property holds for every $i \in \{1, \ldots, n-1\}$. Now we prove it also holds for n.

We now have two cases: n is even and n is odd. We focus only on the former case because the proof for the latter is analogous. Because n is even,

$$F_n = 2F_{n-1} - 1. \tag{8}$$

Observe further, that from Equation (8) we obtain:

$$(F_n + 1)/2 = F_{n-1} \tag{9}$$

Also, observe this means that $(F_n + 1)/2$ is odd, and thus $(F_n - 1)/2$ is even.

Now, by Equation (8),

$$\frac{F_n - 1}{4} = \frac{F_{n-1} - 1}{2}, \tag{10}$$

then we use the definition of F_i to write,

$$\frac{F_n - 1}{4} = \frac{2F_{n-2} + 1 - 1}{2} = F_{n-2}. \tag{11}$$

Substituting with Equations 9 and 10 in Equation (6) we can write:

$$\begin{aligned} \mathcal{Q}_n^{F_n} &= \mathcal{Q}_{n-1}^{(F_n+1)/2} + \mathcal{Q}_{n-2}^{(F_n-1)/4} \\ &= \mathcal{Q}_{n-1}^{F_{n-1}} + \mathcal{Q}_{n-2}^{F_{n-2}}. \end{aligned} \tag{12}$$

Finally, we obtain the desired result by using the inductive hypothesis. ■

Our definition for $\mathcal{N}_{2^k}$ is related to the *Farey Series* (Hardy & Wright, 2008), where the Farey Series of order n, $\mathcal{F}_n$, is the sequence of irreducible fractions between 0 and 1 whose denominators do not exceed n. For example $\mathcal{F}_3$ is equal to $\frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}$. A property of this series is that if a/b, a'/b', and a''/b'' are three consecutive Farey fractions, then $a' = a + a''$ and $b' = b + b''$ (Theorem 29; Hardy & Wright, 2008).

4. A Distance for the 2^k-Neighborhood

In grid path planning it is essential to use informative heuristics. In 4- and 8- neighbor grids, the Manhattan and the Octile Distance, respectively, outperform the Euclidean distance in terms of the number of nodes expanded. Both the Manhattan and the Octile distance correspond to the cost of a shortest path between an arbitrary location and the goal location, ignoring any obstacles.

In this section we develop an analogue of the Manhattan and Octile distances for the more general 2^k-neighborhoods. We focus on answering the question: given an arbitrary i, and two non-negative numbers, x and y, what is the cost of the shortest path between $(0, 0)$ and (x, y) when only moves in $\mathcal{Q}_i$ can be applied?

The problem can be formalized as an Integer Program (IP). Indeed, if $\mathcal{Q}_i = \langle \mathbf{v}_0, \ldots, \mathbf{v}_n \rangle$, we want to solve the following IP, $\mathcal{P}^i$:

$$\text{Minimize } \sum_{i=0}^{n} \alpha_i \|\mathbf{v}_i\| \tag{13}$$

subject to:

$$\sum_{i=0}^{n} \alpha_i \mathbf{v}_i = (x, y), \tag{14}$$

$$\alpha_i \geq 0, \tag{15}$$

for every $i \in \{0, \ldots, n\}$, where each α_i is an integer variable that intuitively represents how many times move $\mathbf{v}_i$ is applied.

Below we prove that the linear-programming (LP) relaxation of $\mathcal{P}^i$ always has an *integer* solution. Before proving such a result, we turn our attention to the LP relaxation of $\mathcal{P}^i$, that we denote $\mathcal{P}^i_{\text{LP}}$, studying its properties and solution.

4.1 A Solution to $\mathcal{P}^i_{\text{LP}}$

Our first result establishes that we can focus on a simpler, two-variable problem instead of the original n-variable problem.

Theorem 7 *Assume $\mathcal{Q}_i = \langle \mathbf{v}_0, \ldots, \mathbf{v}_n \rangle$, and let j be such that (x, y) is contained by $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$. Then $\mathcal{P}^i_{LP}$ is equivalent to $\hat{\mathcal{P}}^i_{LP}$, defined as:*

$$\text{Minimize } \alpha_j \|\mathbf{v}_j\| + \alpha_{j+1} \|\mathbf{v}_{j+1}\| \tag{16}$$

subject to:

$$\alpha_j \mathbf{v}_j + \alpha_{j+1} \mathbf{v}_{j+1} = (x, y), \tag{17}$$

Proof: For the rest of the proof, given an assignment σ of values for every variable α_i in $\mathcal{P}^i_{\text{LP}}$ we denote by $D(\sigma)$ the value of $\sum_{i=0}^{n} \alpha_i \|\mathbf{v}_i\|$. Assume that the optimal solution to $\mathcal{P}^i_{\text{LP}}$ is an assignment σ, to all variables α_i such that $\sigma(\alpha_k) > 0$, for some k such that $k < j$. (The same proof below can be slightly modified to accommodate the case in which $k > j + 1$, but we omit this for simplicity.) Now consider the curve formed by putting all vectors $\mathbf{v}_i$ one after the other, with $\mathbf{v}_k$ put first. Because the curve ends in (x, y) and starts in $(0, 0)$, it must intersect the ray generated by vector $\mathbf{v}_j$. This curve can then be "split" into two parts: the part that goes before such an intersection, and the part that goes after the intersection.

Now we formalize the fact that the sum can be separated in two parts, one containing the vectors before the intersection; the other, containing the vectors coming after. This means (x, y) can be expressed as the sum of vectors whose indices are in two disjoint sets: A^- and A^+, containing, respectively, the indices of the vectors before and after the intersection. The index T, moreover, is used to denote the vector that actually intersects the ray. T does

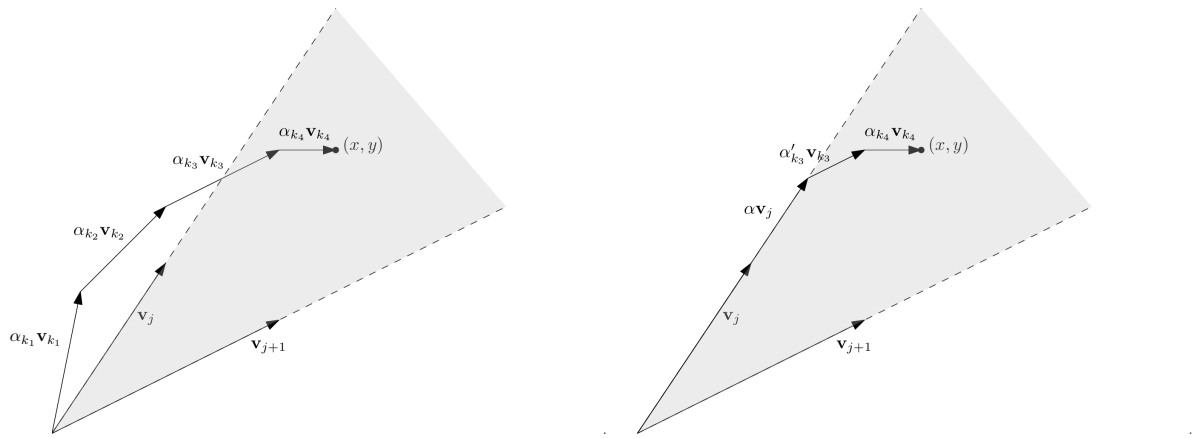


Figure 3: An illustration of the proof of Theorem 7. In this illustration we assume $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$ are consecutive in $\mathcal{Q}$ and (x, y) is contained by $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$. (a) We assume the optimal solution uses vectors $\mathbf{v}_{k_1}, \ldots, \mathbf{v}_{k_4}$. In the proof, this means $A^- = \{k_1, k_2\}$, $T = k_3$, and $A^+ = \{k_4\}$. We observe that the portion of the optimal solution curve to the left of $\mathbf{v}_j$ in (a) can be "replaced" by $\alpha\mathbf{v}_j$, for some α, producing a better solution (b).

not belong to A^- or A^+. Note that A^- is non-empty because it contains k. Finally, for some positive value m and some non-negative value $\beta \leq \alpha_T$ we have that:

$$\sum_{\ell \in A^-} \alpha_\ell \mathbf{v}_\ell + \beta \mathbf{v}_T = m\mathbf{v}_j, \quad \text{for some } m, \text{ and,} \tag{18}$$

$$m\mathbf{v}_j + (\alpha_T - \beta)\mathbf{v}_T + \sum_{\ell \in A^+} \alpha_\ell \mathbf{v}_\ell = (x, y). \tag{19}$$

For an illustration of the previous two equations, see Figure 3.

Note that Equation 19 yields a different assignment, say σ'', that satisfies all constraints of $\mathcal{P}^i_{\text{LP}}$ and which assigns to 0 every α_i such that $i \in A^-$. Furthermore $D(\sigma') < D(\sigma)$. Indeed, this new assignment has replaced a curve of the original solution by a *straight line*.

The argument above can be used with any assignment that uses vectors whose index is either lower than j or greater than $j + 1$. We conclude therefore that the optimal solution must be such that $\alpha_i = 0$, for every i such that $0 \leq i < j$ or such that $j + 1 < i \leq n$. Therefore, the solution to $\mathcal{P}^i_{\text{LP}}$ is equivalent to that of $\hat{\mathcal{P}}^i_{\text{LP}}$.

Finally, observe that $\hat{\mathcal{P}}^i_{\text{LP}}$ has only one feasible assignment yielded by the solution to Equation (17), which must be such that both α_j and α_{j+1} be positive. ■

Given Theorem 7, a solution to $\mathcal{P}^i_{\text{LP}}$ can be computed very easily since $\hat{\mathcal{P}}^i_{\text{LP}}$ has *only one* feasible point. Specifically, this is the assignment to α_j and α_{j+1} that solves the system of two linear equations of Equation (17). Observe also that the solution to $\mathcal{P}^i_{\text{LP}}$ can therefore be computed in constant time.

The following result finally establishes that a solution to $\hat{\mathcal{P}}^i_{\text{LP}}$ has an integer solution, for every $i \geq 0$, and therefore that it is a solution for $\mathcal{P}^i$.

Theorem 8 *$\hat{\mathcal{P}}^i_{LP}$ has a unique integer optimal solution, for every $i \geq 0$.*

Proof: We prove that the values for α_j and α_{j+1} that satisfy Equation (17) are integer, for every i. The proof is by induction on i. For the base case, observe that for $i = 0$ the solution is $\alpha_j = x$ and $\alpha_{j+1} = y$, and thus integer.

Now we assume that the solution for $\hat{\mathcal{P}}^k_{\text{LP}}$ is integer. Let us assume that Equation (17) for $\hat{\mathcal{P}}^{k+1}_{\text{LP}}$ is:

$$\alpha_j \mathbf{v}_j + \alpha_{j+1}\mathbf{v}_{j+1} = (x, y), \tag{20}$$

Because $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$ are consecutive pairs in $\mathcal{Q}_{k+1}$, one of them is in $\mathcal{Q}_k$ whereas the other is the sum of two elements in $\mathcal{Q}_k$. Without loss of generality, let us assume the former is $\mathbf{v}_{j+1}$ and that the latter is $\mathbf{v}_j$. Then we can rewrite Equation (20) as $\alpha_j \mathbf{v}_{j-1} + \alpha_j \mathbf{v}_{j+1} + \alpha_{j+1}\mathbf{v}_{j+1} = (x, y)$, or equivalently:

$$\alpha_j \mathbf{v}_{j-1} + (\alpha_j + \alpha_{j+1})\mathbf{v}_{j+1} = (x, y), \tag{21}$$

where $\mathbf{v}_{j-1}, \mathbf{v}_{j+1} \in \mathcal{Q}_k$. Now we use the inductive hypothesis to conclude that the system of equations given by Equation (21) has one integer solution; hence α_j and $\alpha_j + \alpha_{j+1}$ are integer, which ultimately implies α_{j+1} is integer too. ■

The following result is straightforward from Theorem 8, and essentially says that, to find an optimal path to (x, y), we just need to pick the "closest" moves in the neighborhood, and combine both. We infer our heuristics and canonical orderings from there.

Corollary 9 *Assume $\mathcal{Q}_{k-2} = \langle \mathbf{v}_0, \ldots, \mathbf{v}_m \rangle$, and that cell (x, y) is in the positive quadrant. Furthermore, let j be such that (x, y) is contained by $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$. Then (x, y) can be reached optimally from $(0, 0)$ over $\mathcal{N}_{2^k}$ using an integer combination of $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$.*

4.2 A Heuristic for $\mathcal{N}_{2^k}$

From Theorems 7 and 8, we obtain the following algorithm that computes the length of the optimal path from $(0, 0)$ to a point (x, y) over $\mathcal{N}_{2^k}$, assuming $\mathcal{N}_{2^k} = \langle \mathbf{v}_0, \ldots, \mathbf{v}_m \rangle$:

1. Determine a j such that (x, y) is contained by $\mathbf{v}_j$ and $\mathbf{v}_{j+1}$. To search for such a j we can do a sequential search or a binary search.

2. Solve the system of two linear equations given by Equation (20), and return $\alpha_j \|\mathbf{v}_j\| + \alpha_{j+1}\|\mathbf{v}_{j+1}\|$.

If P is a path planning problem with goal vertex $\mathbf{g}$, and $\mathbf{c}$ is a vertex in the search graph of P, we denote by $h_{2^k}(\mathbf{c})$ the value returned by the procedure above over $\mathbf{g} - \mathbf{c}$.

Theorem 10 *h_{2^k} is a consistent heuristic for $\mathcal{N}_{2^k}$.*

Proof: Let s' be a successor of s then $h_{2^k}(s) \leq c(s, s') + h_{2^k}(s')$ is straightforward from the fact that h_{2^k} is a distance function over an obstacle-free grid. ■

Given k, it is not hard to generate pseudocode that will carry out the necessary computation to complete steps 1 and 2. Algorithm 1 shows the resulting heuristics for different values of k. In them, Step 1 is resolved using sequential search, and thus, in the worst case is needs to perform $O(2^{k-2})$ checks.

Algorithm 1: Heuristics for the 8-, 16-, 32-, and 64-neighborhoods

```
1  function h_8(x, y)
2      if x > y then swap x and y
3      return (y − x) + √2 x
4  function h_16(x, y)
5      if x > y then swap x and y
6      if 2x < y then return (y − 2x) + √5 x
7      else return √5(y − x) + √2(2x − y)
8  function h_32(x, y)
9      if x > y then swap x and y
10     if 3x < y then return (y − 3x) + √10 x
11     else if 2x < y then return √10(y − 2x) + √5(3x − y)
12     else if 3x < 2y then return √5(2y − 3x) + √13(2x − y)
13     else return √13(y − x) + √2(3x − 2y)
14 function h_64(x, y)
15     if x > y then swap x and y
16     if 4x < y then return (y − 4x) + √17 x
17     else if 3x < y then return √17(y − 3x) + √10(4x − y)
18     else if 5x < 2y then return √10(2y − 5x) + √29(3x − y)
19     else if 2x < y then return √29(y − 2x) + √5(5x − 2y)
20     else if 5x < 3y then return √5(3y − 5x) + √34(2x − y)
21     else if 3x < 2y then return √34(2y − 3x) + √13(5x − 3y)
22     else if 4x < 3y then return √13(3y − 4x) + 5(3x − 2y)
23     else return 5(y − x) + √2(4x − 3y)
```

4.2.1 A 2^k Iterative Heuristic

Solving Step 1 using binary search is also possible. This means that, for a specific k we can write a pseudo code that would need, during execution, $O(k)$ "if" checks in the worst case. Nevertheless, the pseudo-code itself would be *exponential* in k. Interestingly, it is possible to construct a function that receives k as parameter, whose execution time is linear in k, and whose size is constant.

The key observation is that to compute the heuristic it is not necessary to carry out a two-step approach described above in Section 4.2. At an abstract level, the algorithm can be viewed as carrying out the binary search and solving the systems of equations at the same time. The pseudocode is shown in Algorithm 2.

Each loop of the algorithm can be understood as playing a "factorization round". Each factorization round uses two consecutive moves of $\mathcal{N}_{2^k}$. To illustrate this, imagine we want

Algorithm 2: A general distance function for $\mathcal{N}_{2^k}$

```
1  function distance(x, y, k)
2      l ← (1, 0)
3      r ← (0, 1)
4      repeat k − 2 times
5          if x > y then
6              r ← r + l
7              x ← x − y
8          else
9              l ← r + l
10             y ← y − x
11     return x‖l‖ + y‖r‖
```

to compute the 2^k distance to $(10, 8)$ from $(0, 0)$. Initially, we start with the 4-connected neighborhood, and because $(10, 8) = 10(1, 0) + 8(0, 1)$, 10 is the factor associated with the move $\mathbf{l} = (1, 0)$ while 8 is the factor associated with $\mathbf{r} = (0, 1)$. For the first factorization round (i.e., the first iteration of the main loop) we want to express $(10, 8)$ in terms of the $\mathbf{l} + \mathbf{r} = (1, 1)$, because we know such a move appears in the next neighborhood. To do this, we take the minimum between the two factors (in this case, 8) and use it as the factor for $(1, 1)$. To get the factorization right, we observe that we still need to use move $(1, 0)$, and that its factor is $10 - 8 = 2$. Thus at the end of the first factorization round, we have expressed $(10, 8)$ as $2(1, 0) + 8(1, 1)$. In the next round the move to introduce is $(1, 0) + (1, 1) = (2, 1)$, its factor is $\min\{2, 8\} = 2$, and we still need to use move $(1, 1)$ with factor $8-2 = 6$; thus, we have expressed $(10, 8)$ as $2(2, 1)+6(1, 1)$. As we continue iterating, we find new factorization of moves of $\mathcal{N}_{2^k}$, for increasing values of k.

Theorem 11 *Function distance(a, b, k) returns the cost of an optimal path on the $\mathcal{N}_{2^k}$ neighborhood that reaches (a, b) from $(0, 0)$.*

To prove theorem, we first prove the following two lemmata.

Lemma 12 *After p iterations of the loop of Algorithm 2, $\mathbf{l}$ and $\mathbf{r}$ are consecutive moves of $\mathcal{Q}_p$.*

Proof: We prove this by induction on the number of iterations of the loop of Line 4. The base case is trivial since $(0, 1)$ and $(1, 0)$ are consecutive moves of $\mathcal{Q}_0$. Now we assume that after n iterations $\mathbf{l}$ and $\mathbf{r}$ are two consecutive moves of Q_n. Then there are two cases. First, the condition of the *if* statement of Line 5 is true, then in iteration $n + 1$, $\mathbf{r}$ and $\mathbf{l}$ are consecutive moves in $\mathcal{Q}_{n+1}$ by definition of $\mathcal{Q}$. The proof for the remaining case is analogous. ■

Lemma 13 *The relations $(a, b) = x\mathbf{l} + y\mathbf{r}$, $x \geq 0$, and $y \geq 0$ are invariants of the loop of a call to distance(a, b, k).*

Proof: We also prove it by induction. After zero iterations the three relations hold (recall that the input is such that (x, y) is in the first quadrant). Now we assume that after n

iterations $(a, b) = x\mathbf{l} + y\mathbf{r}$. Then,

$$(a, b) = (x - z)\mathbf{l} + (y - z)\mathbf{r} + z(\mathbf{l} + \mathbf{r}), \tag{22}$$

for any z. In particular, the algorithm behaves as if $z = \min\{x, y\}$. Indeed, if $x > y$, then x is redefined as $x - \min\{x, y\}$, while y is not changed. Otherwise, y is redefined as $y - \min\{x, y\}$. This guarantees that $x \geq 0$ and that $y \geq 0$. Furthermore, Equation (22) plus the way $\mathbf{l}$ and $\mathbf{r}$ are updated guarantee that $(a, b) = x\mathbf{l} + y\mathbf{r}$ holds in iteration $n + 1$. ■

Now we are ready to provide a complete proof for Theorem 11.

Proof (of Theorem 11) : From Lemma 13, it holds that (a, b) is contained by $\mathbf{l}$ and $\mathbf{r}$. In addition, at the end of the execution, $(a, b) = x\mathbf{l} + y\mathbf{r}$, where, by Lemma 12, $\mathbf{r}$ and $\mathbf{l}$ are consecutive moves of Q_{k-2}. We observe, thus, that x and y are a solution to the system of equations of Theorem 8, which in turn implies that $x\|\mathbf{l}\| + y\|\mathbf{r}\|$ is the distance to (x, y) from $(0, 0)$ over $\mathcal{N}_{2^k}$, which implies that Algorithm 2 is correct. ■

5. Practical Grid Path Planning with $\mathcal{N}_{2^k}$

Above we have formally characterized $\mathcal{N}_{2^k}$ and given a consistent (and admissible) heuristic which is perfect for obstacle-free grids. Yet, if we plan to use A* for path planning, an important issue of $\mathcal{N}_{2^k}$ is the increase in branching factor, which is exponential with k. By just observing this fact one might, at first sight, discard an A* implementation of 2^k-neighborhoods. Indeed, with each A* expansion we would need to generate 2^k successors, many of which potentially have to be added to the Open list, incurring in much overhead.

Nevertheless an opposing observation is that in obstacle-free 2^k-grids, the number of optimal paths between two vertices may *decrease substantially* as k increases. This is because, in obstacle-free grids, any reordering of the moves of an optimal path is also an optimal path. With 2^k-neighborhoods, optimal paths may use longer moves, resulting in paths with fewer moves. As an example, let us consider the number of paths between $(0, 0)$ and $(4, 2)$. When $k = 2$ (4-neighborhood), we reach $(4, 2)$ with any sequence of moves that has 4 vertical moves and 2 horizontal moves, yielding $\frac{6!}{4!2!} = 15$ paths. When $k = 3$ (8-neighborhood), we require 2 diagonal (1,1) moves plus 2 vertical moves, yielding $\frac{4!}{2!2!} = 6$ optimal paths. Finally, when $k \geq 4$ there is only one optimal path with two $(2, 1)$ moves.

Thus even though the branching factor increases exponentially with k, the number of optimal paths between two points may decrease substantially with k. Therefore, at least at a theoretical level, it is not obvious that increasing k should degrade performance of grid path planning.

In this section, we show how recent developments in 8-neighbor grid path planning can be leveraged for 2^k-neighborhoods. Specifically, we show how Jump Point Search (JPS) and Canonical A* can be adapted to the $\mathcal{N}_{2^k}$ neighborhood. Both JPS and Canonical A* have been shown to improve the performance of a regular A* on the 8- and 4-neighbor grids, and thus understanding how to incorporate it into the 2^k is important since we would also expect to see performance improvements compared to a regular A* implementation.

We follow the view of JPS given by Sturtevant and Rabin (2016), who show that JPS can be understood as the composition of two ideas: a canonical orderings over paths of the grid, and a specific successor function.

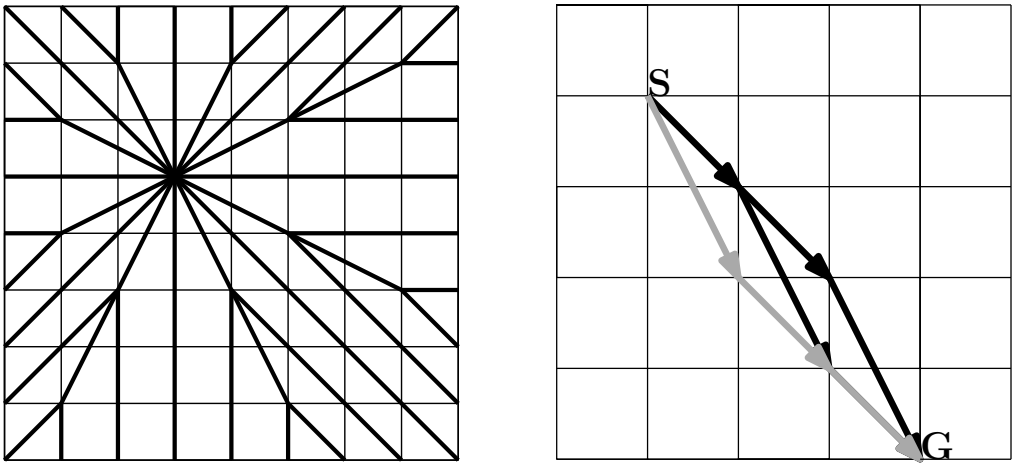


Figure 4: Left: An illustration of the canonical paths on the 16-neighborhood. Right: The canonical path (in gray) and two non-canonical paths (in black) between S and G.

5.1 Canonical Paths for $\mathcal{N}_{2^k}$

Following the definitions by Sturtevant and Rabin (2016), a *canonical path* on an 8-connected grid is one that is generated by using a sequence of moves of the form $d^n c^m$, where d is a diagonal move and c is a cardinal move adjacent to d. The two most important properties of canonical paths are: (i) that there is at most one canonical path between each pair of vertices, and (ii) that every canonical path is also an optimal path. Property (ii) follows from our Corollary 9, and property (i) follows from the restriction on the order of the two moves. (For an illustration of canonical paths on an 8-connected grid, see Figure 5a.)

Adapting this definition to $\mathcal{N}_{2^k}$ is not hard if we guarantee these two properties identified above. From Corollary 9, we know that each vertex in the obstacle-free grid is reached optimally by applying only two consecutive moves from $\mathcal{N}_{2^k}$, for any k. Thus to satisfy condition (ii) we need to focus only on paths with two consecutive moves. Finally, to satisfy condition (i) we can do as with 8-connected grids preferring one of those moves to occur before the other move. In the definition below, we choose odd moves first because this generalizes the definition for canonical path given by Sturtevant and Rabin (2016) for 8-connected grids:

Definition 14 *A path over $\mathcal{N}_{2^k}$ is a* canonical path *iff it is generated by the sequence of moves $\mathbf{v}^n\mathbf{u}^m$, where $\mathbf{v}$ is an odd move in $\mathcal{N}_{2^k}$, $\mathbf{v}$ is either $\mathbf{v}-1$ or $\mathbf{v}+1$, and $n, m \geq 0$.*

As an illustration, Figure 4 shows the canonical paths that originate from one vertex on a 16-connected grid. Note here that $(2,1)$, and its variants $(1,2)$, $(1,-2)$, and so forth, are the only odd moves.

5.2 Canonical A* for $\mathcal{N}_{2^k}$

Canonical A* is a version of A* which, intuitively focuses on building canonical paths to the goal. The advantage that this has in practice is that a single vertex cannot of the search space cannot be rediscovered via another (equally good) path. To this end, Canonical A* uses Harabor and Grastien (2011)'s notion of *natural* and *forced* successors in order to prune the set of successors of a node. Intuitively, by only considering natural successors, A* naturally builds canonical paths only. Nevertheless, because not all pairs of reachable nodes are reachable via a canonical path, it is necessary to define the notion of forced successors, which may be needed when the search expands a vertex that is the convex corner of an obstacle.

Now we provide formal definitions for these two types of successors. As these definitions are used in the context of an execution of Canonical A*, we refer to the concept of expansion within the definitions.

Definition 15 *Given search node* $\mathbf{u}$ *be such that it is expanded from its parent via move* $\mathbf{m}$*. A successor* $\mathbf{v}$ *of* $\mathbf{u}$ *is* natural *over* $\mathcal{N}_{2^k}$ *iff:*

1. *If* $\mathbf{m}$ *is odd, then* $\mathbf{v}$ *is in* $\{\mathbf{u}+(\mathbf{m}-1), \mathbf{u}+\mathbf{m}, \mathbf{u}+(\mathbf{m}+1)\}$.
2. *Otherwise, then* $\mathbf{v} = \mathbf{u}+\mathbf{m}$.

The natural successors of $\mathbf{u}$ *are denoted by natural*$(\mathbf{u})$.

Furthermore, if $\mathbf{v}$ is the root node of the search (and thus has not been expanded from any node), then $Succ_{\mathcal{N}_{2^k}}(\mathbf{v})$ are the natural successors of $\mathbf{v}$.

As mentioned above, when focusing on natural successors, the search will generate canonical paths only. For an illustration, see Figure 5(a). Now we define the notion of forced successors following Harabor and Grastien (2011):

Definition 16 *Given a search node* $\mathbf{u}$ *expanded from its parent* $\mathbf{t}$ *via a move* $\mathbf{m}$ *over neighborhood* $\mathcal{N}_{2^k}$*, a successor* $\mathbf{v}$ *of* $\mathbf{u}$ *is* forced *iff:*

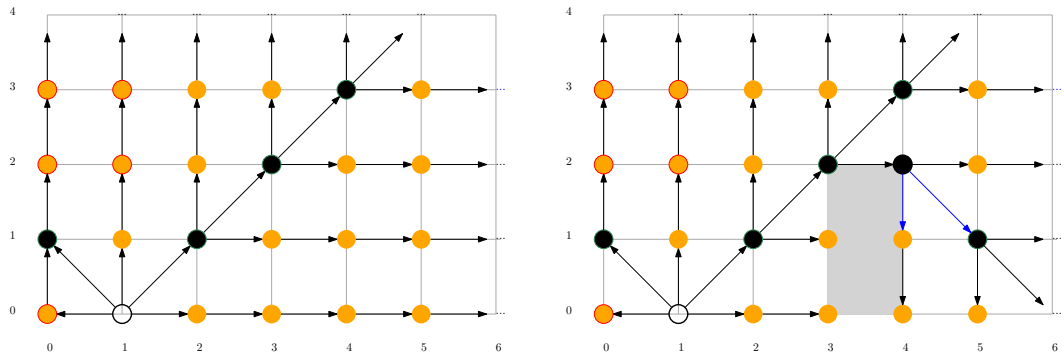
- $\mathbf{v}$ *is not a natural successor of* $\mathbf{u}$ *over* $\mathcal{N}_{2^k}$*, and*
- *path* $\mathbf{tuv}$ *is the only optimal path between* $\mathbf{t}$ *and* $\mathbf{v}$.

The forced successors of $\mathbf{u}$ *are denoted by forced*$(\mathbf{u})$.

Forced successors appear only when expanding vertices which are the convex corner of an obstacle cell.

Now we are ready to describe Canonical A* for $\mathcal{N}_{2^k}$, which generalizes Sturtevant and Rabin (2016)'s Canonical A* for 8-connected grids. Canonical A* can be viewed as an algorithm that, when expanding a node, prunes away all nodes that are not among its natural or forced successors. More specifically, when expanding the root node $\mathbf{r}$, all nodes in $Succ_{\mathcal{N}_{2^k}}(\mathbf{r})$ are added to the open list.

Nodes that have forced successors can only correspond to vertices which are convex corners of an obstacle. Enumerating such forced successors is simple to do. Upon expanding a convex corner of an obstacle $\mathbf{v}$ via move $\mathbf{m}$, the first step intuitive step is to decide where



(a) (b)

Figure 5: Assuming a search is rooted in node $(1, 0)$, black arrows illustrate the natural neighbor relation, while blue arrow illustrate the forced neighbor relation. Bidimensional nodes, which are those nodes expanded via an odd move, are black-filled. Unidimensional nodes, which are those nodes expanded via an even move, are orange-filled. Observe that due to the obstacles, in grid (b) both $(4, 1)$ and $(5, 1)$ are forced successors of $(4, 2)$.

the search should "turn". To determine this, observe that $\mathbf{v}$ has two successors which are on the border of o, one generated by a horizontal move applied to o, and one generated y a vertical move applied to o; let us denote those moves by $\mathbf{m}_h$ and $\mathbf{m}_v$. Now only one among $\mathbf{m}_h$ and $\mathbf{m}_v$ forms a minumum angle with $\mathbf{m}$. Denote such a move by $\mathbf{m}'$. Forced successors are all those successors of $\mathbf{v}$ that are not natural and that are generated by moves between $\mathbf{m}$ and $\mathbf{m}'$. For an illustration, consider the expansion of node $(4, 2)$ in Figure 5(b), which is the convex corner of an obstacle. $\mathbf{m}_h$ and $\mathbf{m}_v$ are, respectively $(-1, 0)$ and $(0, -1)$. $\mathbf{m}_v$ is the move that forms the minimum angle with $\mathbf{m}$, therefore forced successors of $(4, 2)$ are all successors contained within $\mathbf{m}_v$ and the natural successors of $(4, 2)$.

Theorem 17 *Canonical A*, as described above, finds an optimal solution to any path planning problem P over $\mathcal{N}_{2^k}$, for every k.*

The proof for Theorem 17 follows directly from lemmata 19 and 20, which we explain below. The first result is that there always exists an optimal path that is a concatenation of canonical paths. Henceforth we assume that if σ_1 is a path that ends at vertex v and that σ_2 is a path that starts at vertex v, then $\sigma_1 \circ \sigma_2$ denotes the concatenation of σ_1 and σ_2 without repeating v.

Definition 18 *Given a path planning problem P a concatenation of canonical paths over P, $\sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_n$ $(n \geq 1)$ is* irreducible *if $\sigma_i \circ \sigma_{i+1}$ is not a canonical path, for every $i \in \{1, \ldots, n-1\}$.*

Lemma 19 *For any solvable path planning problem P over $\mathcal{N}_{2^k}$, there exists an optimal solution which is either a canonical path or an irreducible concatenation of two or more canonical paths.*

Proof: The main idea of the proof is that any optimal path can be transformed into a concatenation of canonical paths. Intuitively, whenever possible, we can replace a portion of any optimal path by a canonical path, until we can no longer do so. By applying this repeatedly, we end up with a path that has the required form. The function of Algorithm 3 does such a transformation. Given an optimal path σ, *MaximalCanonical*(σ) returns a path that can be expressed as an irreducible concatenation of canonical paths. ■

Algorithm 3: Transforms an optimal path into a concatenation of maximal canonical paths

```
1 function MaximalCanonical(σ)
2     if σ contains one or fewer vertices then
3         return ⟨σ⟩
4     Let σ = c_1 c_2 ... c_n, and let i be the largest number in {2,...,n} such that there exists
        a canonical path between c_1 and c_i
5     Let σ' be the canonical path between c_1 and c_i
6     return ⟨σ'⟩ concatenated with MaximalCanonical(c_i ... c_n)
```

Lemma 20 *Let P be a path planning problem whose optimal solution is an irreducible concatenation of canonical paths $\sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_n$. Then the last vertex of σ_i (and first vertex of σ_{i+1}), for every $i \in \{1, \ldots, n-1\}$, is a convex corner of an obstacle cell.*

Proof: For a contradiction, let us assume that σ_J and σ_{J+1} violate the condition of the lemma. Let us denote the first vertex of σ_J by **a** and the last vertex of σ_{J+1} by **b**.

Figure 6 illustrates, in a specific grid, all possible concatenations of two canonical paths between two points that cannot be connected by a canonical path. Canonical paths that originate in **a** are shown in green, whereas canonical paths that end in **b** are shown in red. Observe that a polygon (shaded in gray) defined by the 'last canonical path that is not 'cut' by the obstacle is defined.[3] All concatenations of canonical paths between **a** and **b** are formed by connecting **a** to a vertex in the shaded area via a canonical path, and then connecting such a vertex with **b**.

We want to show that the shortest path between **a** and **b** is formed when the vertex that connects both canonical paths is precisely the corner of the polygon (which indeed is also the corner of an obstacle area preventing direct connection).

To see why this is true we adapt the proof for the any-angle neighborhood—which is slightly easier to follow—to the 2^k-neighborhood. In the anytime neighborhood, shortest paths are also *taut*. Informally a path is taut when, *"treated as a string, it cannot be made 'tighter' by pulling on its ends"* (Oh & Leong, 2017); in other words, optimal paths 'bend' only at the corners of obstacles. Using the triangular inequality, it is easy to show that taut paths are always shorter than non-taut paths. To see this, consider the example of Figure 7, which shows a taut path between **a** and **b**, a taut path **acb** and a non-taut path **adb**. To prove the taut path is shorter, let **e** be the point at which segment **ad** and the continuation of **cb** to the left intersect. Now we write one of the triangle inequalities associated to the triangle **ebd**,

3. In the general case, indeed, more than one polygon may be generated. This does not make the rest of the proof less general.

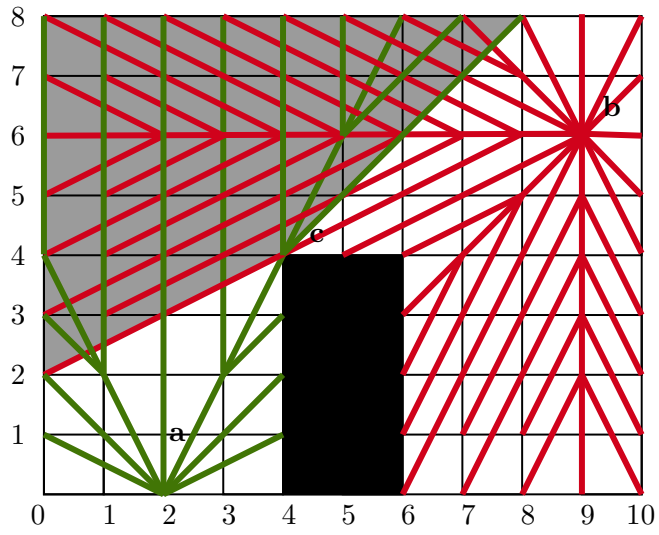


Figure 6: All canonical paths originating in **a** are shown in green while all canonical paths ending in **b** are shown in red. Each vertex in the shaded area defines a concatenation of two canonical paths that reach **b** from **a**.

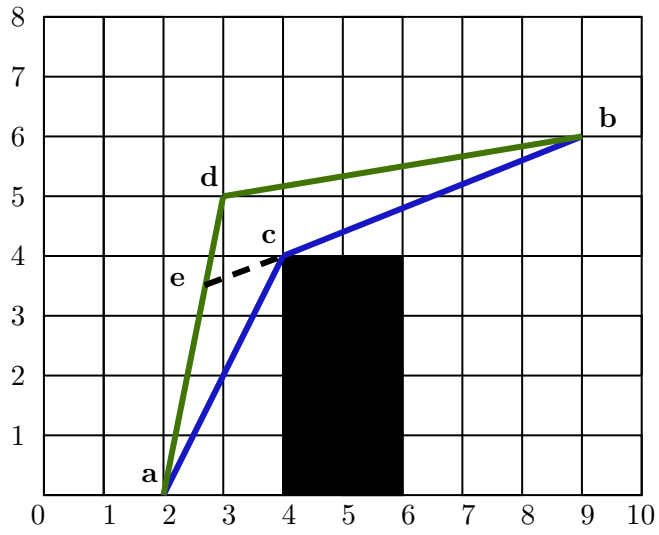


Figure 7: Taut path π_{acb} is shorter than non-taut path π_{adb} for an any-angle neighborhood.

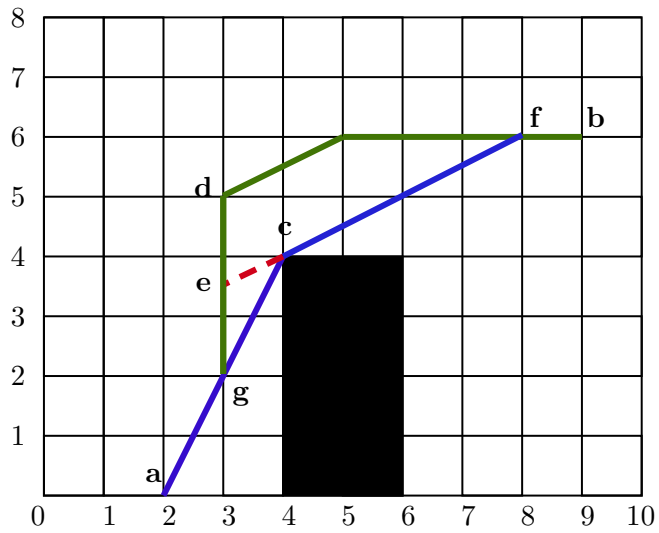


Figure 8: Taut path π_{acb} is shorter than non-taut path π_{adb} on a 16-neighborhood.

$$\|\mathbf{e}-\mathbf{c}\| + \|\mathbf{b}-\mathbf{c}\| \leq \|\mathbf{d}-\mathbf{e}\| + \|\mathbf{b}-\mathbf{d}\| \tag{23}$$

Now we write one of the triangular inequalities for triangle **ace**:

$$\|\mathbf{c}-\mathbf{a}\| \leq \|\mathbf{e}-\mathbf{a}\| + \|\mathbf{e}-\mathbf{c}\| \tag{24}$$

Summing up (23) and (24), we obtain the desired result:

$$c(\mathbf{acb}) = \|\mathbf{b}-\mathbf{c}\| + \|\mathbf{c}-\mathbf{a}\| \leq \|\mathbf{d}-\mathbf{e}\| + \|\mathbf{b}-\mathbf{d}\| + \|\mathbf{e}-\mathbf{a}\| = c(\mathbf{adb}) \tag{25}$$

Note that because the two paths do not coincide completely (25) is actually a strict inequality in practice.

Now we show that the same relation holds in 2^k neighborhoods, this time when we compare a taut concatenation of canonical paths with a non taut one. To see this, observe Figure 8 which shows two 16-neighborhood paths between **a** and **b**. Before we continue with the analysis, we use the notation π_{xyz} to refer to the path shown in the figure, that goes from **x** to **z**, visiting **y**. Furthermore we denote by π_{ec} the segment that connects **e** and **c**, where **e** is the intersection between π_{adb} and the continuation of path π_{cfb} to the left. Note that **e** is a point that may or may not be a vertex (in Figure 8 it is actually not a vertex).

We assume that both π_{acb} and π_{adb} are concatenations of canonical paths; however the former is taut whereas the latter is not. The key to the rest of the proof is that we can write triangular inequalities that are analogous to the any-angle case. Indeed, because of Theorem 7, π_{ecb} is a shortest segment that connects **e** and **b** using 16-neighborhood moves, since it is generated using consecutive moves of the 16-neighborhood. This allows us to write:

$$c(\pi_{ec}) + c(\pi_{cfb}) < c(\pi_{edb}). \tag{26}$$

Observe this is a strict inequality since we assume that $\mathbf{d} \neq \mathbf{c}$.

Finally, note that, because the path π_{agc} is optimal, we can write:

$$c(\pi_{agc}) \leq c(\pi_{ec}) + c(\pi_{age}). \tag{27}$$

Summing up (26) and (27) we obtain $c(\pi_{acb}) < c(\pi_{adb})$. Although we have exemplified this proof using a 16-neighborhood, the same argument can be used for any 2^k-neighborhood.

To complete the proof, we can use a contradiction argument: if σ_J and σ_{J+1} are such that the last element of σ_J is not a corner, then there must be another path $\sigma'_J \circ \sigma'_{J+1}$ that does bend on a corner, and $\sigma'_i \circ \sigma'_{i+1}$ should be shorter than $\sigma_i \circ \sigma_{i+1}$ (and this is shown using the argument above), which contradicts the fact that the original path is optimal. ■

Proof (of Theorem 17) : From the analysis above, we observe that if the problem has a solution, then there is a concatenation of canonical paths that reaches the goal optimally. Furthermore, from Lemma 20, we know that the vertex visited in between such canonical paths then such a vertex is also a corner of an obstacle. Because Canonical A* fully expands every convex corner of an obstacle, each of the canonical paths of the partition can be found by Canonical A*. We conclude that Canonical A* is complete and optimal. ■

5.3 Jump Point Search

Jump Point Search (JPS) (Harabor & Grastien, 2011) is a search strategy in which, like in Canonical A*, upon expanding a node, we only compute the natural and forced successors. However, instead of immediately adding these successors directly to an Open list, we compute a so-called *jump point* for each of them, which we then may add to the Open list. More precisely, the way we compute the successors to add to the open list, we borrow the successor generator described by Harabor and Grastien (2011), in Algorithm 4. In

Algorithm 4: JPS successor generator

```
   Input: v, current node, g: goal node
1  successors ← ∅
2  for each n ∈ natural(v) ∪ forced(v) do
3      n ← jump(v, direction(v, n), g)
4      if n ≠ null then
5          successors ← successors ∪ {n}
6      return successors
```

the pseudo-code, the call $jump(\mathbf{v}, direction(\mathbf{v}, \mathbf{n}), g)$ computes the jump point from $\mathbf{v}$ in direction $direction(\mathbf{v}, \mathbf{n})$. Finally, $direction(\mathbf{v}, \mathbf{n})$ denotes the move that generates $\mathbf{n}$ from $\mathbf{v}$.

The jump point of a node $\mathbf{v}$ in direction $\mathbf{d}$ intuitively corresponds to the node that is found by trying by advancing in direction $\mathbf{d}$ and stoping when we have reached the goal or a node with forced successors. In addition, if $\mathbf{d}$ is an odd move, we also stop when we have found a node from which there is a jump point in any of the two directions that are immediately adjacent to $\mathbf{d}$. The precise definition is given below.

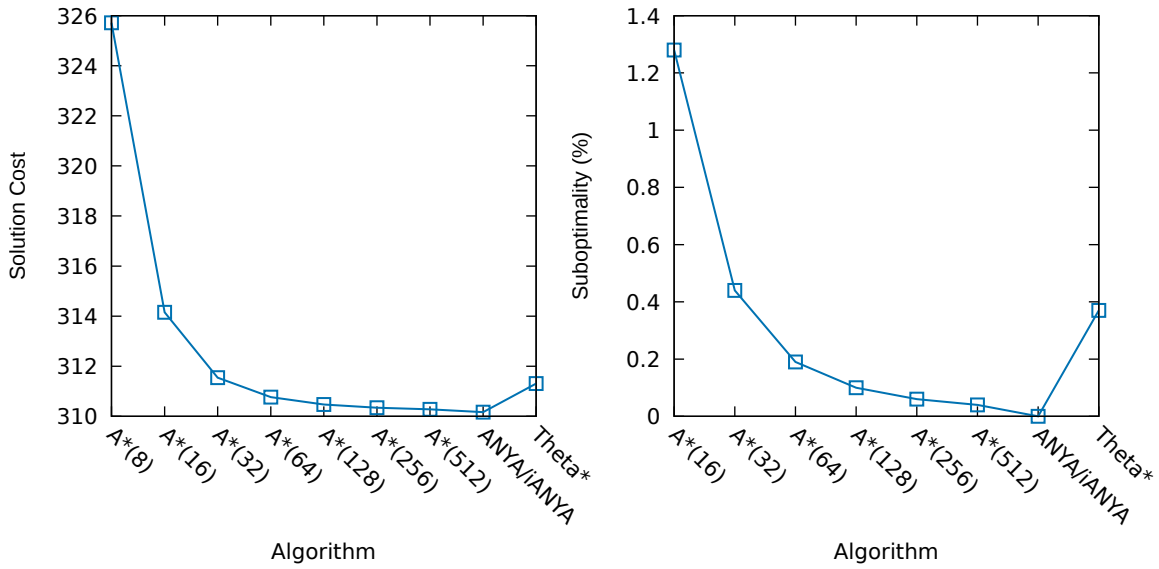


Figure 9: Game Maps: Average solution cost and suboptimality per algorithm.

Definition 21 (2^k Jump Point) *A node* $\mathbf{j}$ *is the jump point of* $\mathbf{v}$ *in direction* $\mathbf{d}$ *in* $\mathcal{N}_{2^k}$ *if* $\mathbf{j} = \mathbf{v} + k\mathbf{d}$ *and* k *is the minimum natural number greater than 0 such that:*

1. $\mathbf{j}$ *is the goal, or*
2. $\mathbf{j}$ *has forced successors, or*
3. *if* $\mathbf{d}$ *is an odd move then there exists a jump point of* $\mathbf{j}$ *in direction* $\mathbf{d}+1$ *or* $\mathbf{d}-1$.

Computing jump points is quite simple and can be done by adapting the function of Harabor and Grastien (2011), modified as in Algorithm 5.

Algorithm 5: *jump* function

```
   Input: v, initial node, d: direction,g: goal node
1  n ← v + d
2  if n is not a vertex then
3      return null
4  if n = g then
5      return n
6  if n has forced successors then
7      return n
8  if d is an odd move then
9      if jump(v, d + 1, g) ≠ null or jump(v, d − 1, g) ≠ null then
10         return n
```

6. Empirical Findings

The objective of our evaluation was twofold. First, we wanted to investigate the impact on solution quality of using $\mathcal{N}_{2^k}$ with A*, the most standard heuristic search algorithm, and to

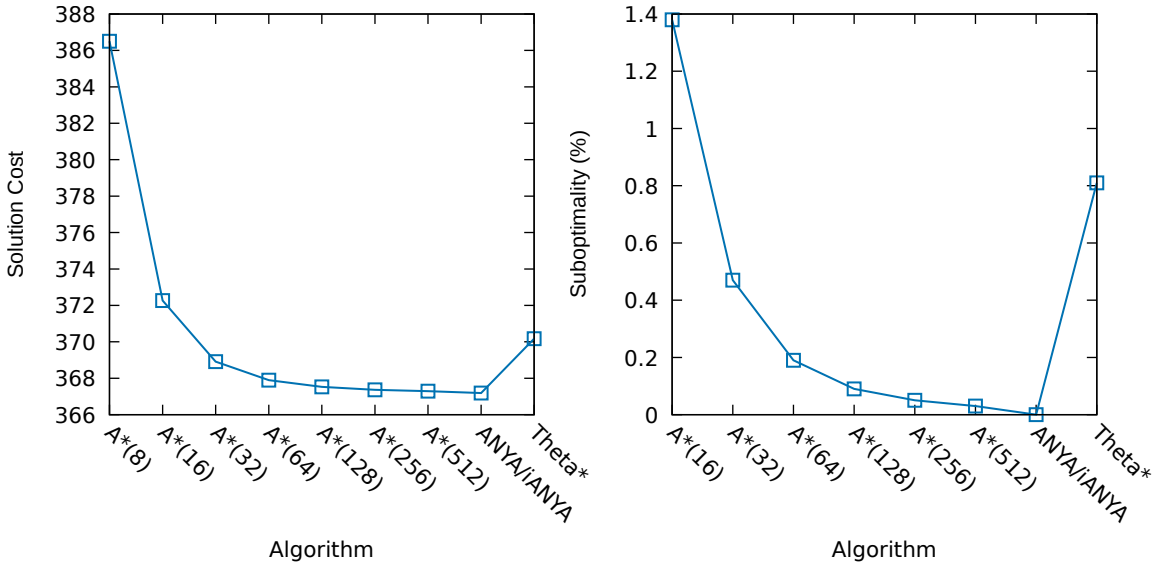


Figure 10: Room Maps: Average solution cost and suboptimality per algorithm.

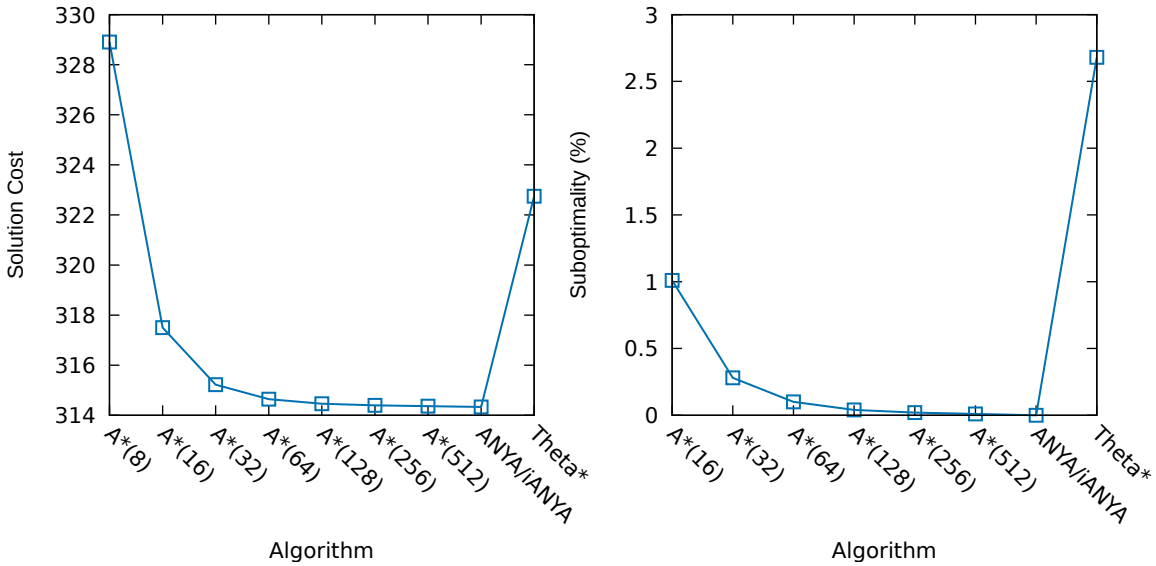


Figure 11: Random Maps: Average solution cost and suboptimality per algorithm.

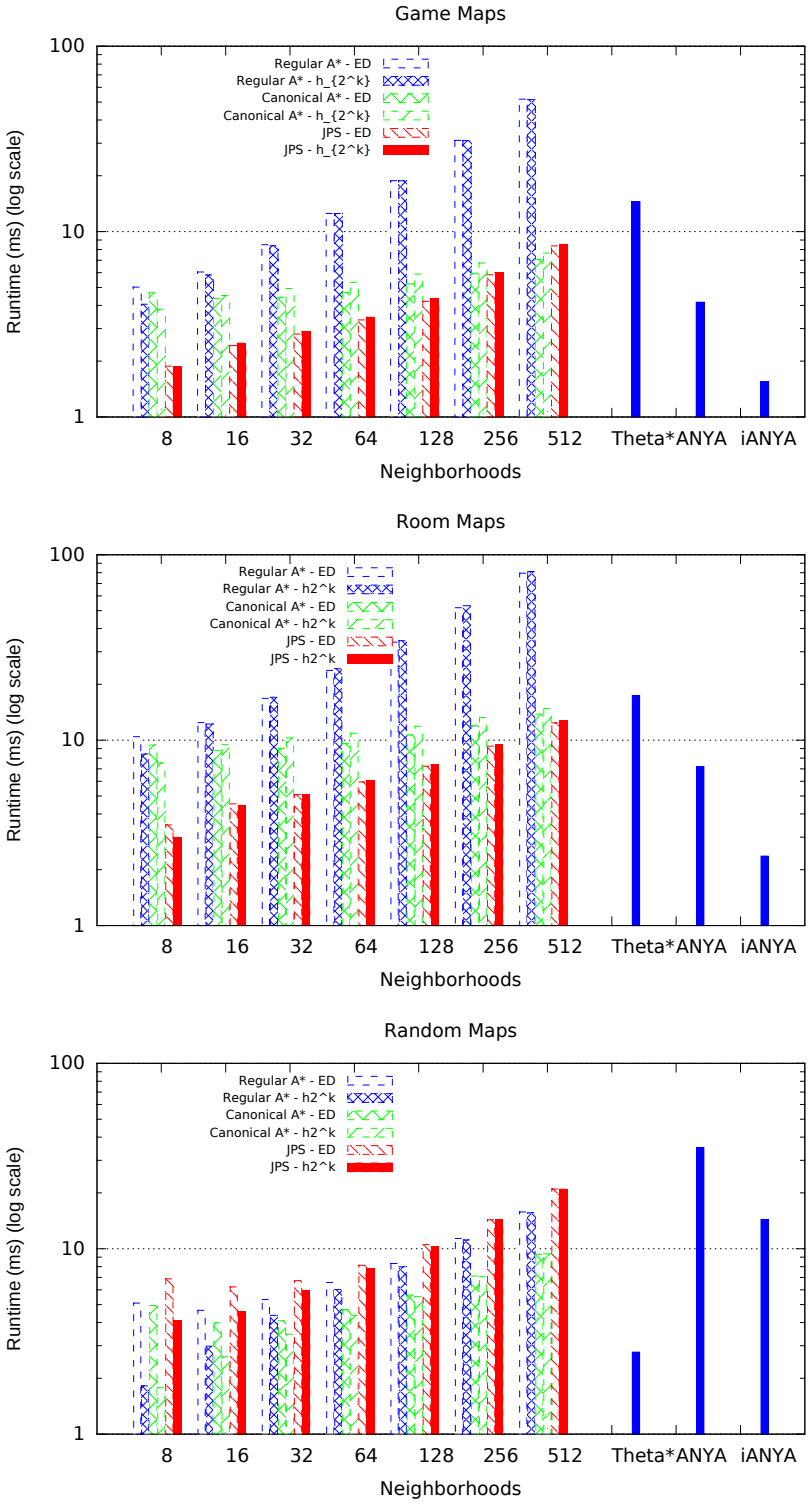


Figure 12: Average runtime per neighborhoods.

compare the obtained solution with those generated by the any-angle path planners ANYA and Theta*. Second, we wanted to investigate the impact that increasing k had on the runtime performance of A*, Canonical A*, and JPS using h_{2^k} heuristic and the Euclidean distance. In addition, we compare the runtime performance with ANYA and Theta*.

We implemented our algorithms on top of Uras and Koenig's implementation of Subgoal graphs (2015), which uses a standard binary heap for *Open*. The implementation includes Theta* and a version of the any-angle planner described by Harabor and Grastien (2013), which below we call ANYA. We also include the results of an optimized version of the same any-angle planner, described by Harabor et al. (2016), which we call iANYA below except for random maps.[4] Implementation details can be found in Uras and Koenig. All experiments were run on a 2.20GHz Intel(R) Xeon(R) CPU Linux machine with 128GB of RAM.

Our implementation of JPS makes two modifications to the original proposal which makes the implementation easier while preserving all properties of the original JPS. First, jump points are either the goal or convex corners of obstacles; second, jump points, when expanded are fully expanded, that is, all its successors are added to the open list.

In our comparison, we use three sets of game maps from the MovingAI repository (Sturtevant, 2012). The first set, game maps, are from *Baldurs Gate II*—which contains maps of size 512×512—, from *Dragon Age: Origins of Size*—which range from 22×28 to 1260×1104—, and from *StarCraft*—which range from 384×384 to 1024×1024. The second set, are random maps of size 512×512, where the percentage of blocked cells varies from 10% to 40%. Finally, the third set are room maps of size 512×512, where the room size varies from 8×8 to 64×64.

Figures 9–11 show the average solution cost of Regular A* for different neighborhoods, ANYA, iANYA, and Theta*. These figures also show the average percentage suboptimality, omitting A*($k = 8$) to obtain better plot scaling. We evaluated seven values for 2^k: 8, 16, 32, 64, 128, 256, and 512. We make the following observations.

- Solution cost improves when k increases in all three benchmarks. Improvements are marginal as k increases over 6 (64-neighborhood).
- Compared with ANYA (and iANYA), an optimal any-angle planner, we observe that A* obtains almost optimal any-angle paths when k over 6 (64-neighborhood). Specifically, over the 64-neighborhood, the suboptimality is only 0.19%, 0.19%, and 0.10% on the Games, Rooms, and Random maps, respectively.
- Compared to Theta*, we observe that A* with k greater than 4 (16-neighborhood) finds better solutions in Game Maps and Room Maps. In Random Maps, with k greater than 3 (8-neighborhood) A* obtains a better average solution cost.

Now we turn our attention to our runtime evaluation. In our implementation, Canonical A* and JPS did not use any pre-computation or other optimizations such as the ones described by Rabin and Sturtevant (2016) or Harabor and Grastien (2014). We consider the implementation of the optimization methods part of the future work.

4. It is important to note that in ANYA the agent is placed in the center of cells, thus we include it here only as a reference.

In order to understand the effects over runtime of the different algorithms, we present Figure 12, which shows the average runtime of Regular A*, Canonical A* and JPS for different neighborhoods. We evaluated h_{2^k} heuristic and the Euclidean distance (ED) for each algorithm. We evaluated the same seven values for k. We make the following observations.

- **On Regular A***. Runtime increases with k on the Game and Room maps. This can be explained by the larger branching factor, which increases exponentially with k. A* runs faster using the h_{2^k} heuristic than using the Euclidean distance for small values for k. For intermediate and higher values for k, A*, used with ED, is slightly faster due to the overhead in computing our heuristic and because h_{2^k}, when k is large, tends to be more similar to the Euclidean distance. In Random maps, A* with h_{2^k} is faster than A* with the Euclidean distance for every k. In Random maps, h_{2^k} is a good heuristic, especially when there is a small percentage of obstacles.

- **On Canonical A***. The main observation is that Canonical A* with Euclidean distance has better performance for most values of k in Game and Room maps. In Random maps h_{2^k} is the best option for most values for k.

- **On Jump Point Search**. JPS with the Euclidean distance is the best option for most values for k in Game maps (except for $k = 3$), though it is outperformed by ANYA for $k > 6$, and by iANYA in all configurations except for Random maps. In Room maps, the Euclidean distance obtains best performance for $k = 3$ and $k = 4$, for other values, JPS with ED is slightly faster. In Random maps the h_{2^k} heuristic is the best option.

- **On all algorithms**. JPS obtains the best performance in Game Maps and Room Maps (except for $k = 9$ in Game Maps). On the other hand, JPS obtains the worst performance in Random maps. This can be explained because JPS, when expanding a state, generates all states on canonical paths that emerge from such a state Sturtevant and Rabin (2016).

The optimized iANYA planner outperforms our algorithms even in the basic configutation ($k = 2$), except in Random Maps. The runtime of Theta* is 14.49 ms, 17.39 ms and 2.77 ms in Game, Room, and Random maps, respectively. Canonical A* and JPS using the 2^k neighborhood are faster than Theta*, for every k in Game and Room maps. Furthermore, better solution costs are obtained, as shown in Figures 9-11. In Random maps, Canonical A* with the ED heuristic using $k = 4$ is faster than Theta* and obtains better cost, with higher k the Theta* is faster, however is worse in solution cost. The runtime of ANYA is 4.16 ms, 7.20 ms and 35.03 ms in Game, Room, and Random maps, respectively. In Game and Room maps, JPS is faster than ANYA until k is equal to 6 (64-neighborhood). With $k = 7$ (128-neighborhood), runtimes are similar. For $k > 7$ JPS (without optimizations) can be faster than ANYA and competitive in the solution quality. In Random maps, Regular A*, Canonical A* and JPS using 2^k-neighbor are faster than ANYA for all values of k that we evaluated.

Additionally, we include four tables that show the number of expansions, heap percolations and the time per expansion. Table 1 shows the results for Regular A*. We observe

Table 1: Expansions, percolations and time per expansion in milliseconds for Regular A*.

Game Maps						
	h_{2^k} heuristic			Euclidean heuristic		
Neigh	Exp	Perc	Time/Exp	Exp	Perc	Time/Exp
8	18,397	186,688	0.23	23,382	192,285	0.22
16	18,638	195,007	0.32	20,461	194,805	0.30
32	18,731	203,152	0.45	19,537	202,358	0.44
64	18,761	214,690	0.66	19,179	213,933	0.64
128	18,770	227,334	1.00	19,015	226,644	0.98
256	18,772	241,090	1.66	18,929	240,603	1.63
512	18,773	255,490	2.76	18,879	255,212	2.73
Random Maps						
8	7,152	96,647	0.26	21,879	200,275	0.24
16	8,131	108,067	0.37	13,600	146,221	0.34
32	8,796	116,263	0.50	11,024	130,960	0.48
64	9,121	122,119	0.67	10,142	128,500	0.66
128	9,273	126,752	0.87	9,810	130,078	0.86
256	9,355	130,513	1.20	9,669	132,429	1.20
512	9,402	133,396	1.67	9,600	134,590	1.66
Room Maps						
8	34,611	396,221	0.25	43,253	398,015	0.24
16	35,219	408,918	0.35	37,409	394,920	0.33
32	35,204	419,189	0.49	35,920	408,311	0.48
64	35,172	436,346	0.69	35,467	429,116	0.68
128	35,155	454,675	0.99	35,301	449,673	0.96
256	35,150	473,379	1.51	35,232	469,872	1.48
512	35,147	490,075	2.32	35,198	487,452	2.29

that the search time per expansion increases when the neighborhood size increases. In addition, we observe a smooth increase of the search time per expansion when Canonical A* is used, as Table 2 shows. Table 3 shows the results for JPS. The number of expansions correspond to the number of canonical expansions performed in a breadth first search fashion in order to identify the Jump Points that are inserted in the *Open* list. This explains the large number of expansions and the small number of percolations. Here, the search time per expansion is small, but it increases when the neighborhood increases.

Table 4 shows the results for Theta* and ANYA. The explanation of what is an expansion in Theta* and ANYA can be found in the paper that explaining the implementation that we use Uras and Koenig. Due to the way the algorithms work, Theta* and ANYA have a small overhead per expansion compared to Regular A* with small k. For big k values, Regular A* has a larger overhead. On the other hand, Canonical A* and JPS seem like better options if time per search expansion and total search time are considered. A general observation for results in Table 1, 2 and 3 is that the h_{2^k} heuristic allows to obtain better results than the Euclidean heuristic for small k values.

Table 2: Expansions, percolations and time per expansion in milliseconds for Canonical A*.

Game Maps						
	h_{2^k} heuristic			Euclidean heuristic		
Neigh	Exp	Perc	Time/Exp	Exp	Perc	Time/Exp
8	18,399	178,463	0.20	23,413	174,094	0.20
16	18,641	161,476	0.24	20,488	153,831	0.21
32	18,733	157,899	0.26	19,559	153,336	0.22
64	18,762	163,643	0.28	19,197	161,175	0.24
128	18,771	176,693	0.31	19,030	175,451	0.27
256	18,773	196,521	0.36	18,941	195,982	0.31
512	18,773	222,243	0.40	18,890	222,040	0.37
Random Maps						
8	7,152	94,883	0.25	21,880	191,866	0.22
16	8,132	105,373	0.32	13,601	140,632	0.29
32	8,796	113,265	0.39	11,025	127,255	0.37
64	9,121	118,929	0.48	10,142	125,349	0.46
128	9,273	123,360	0.59	9,811	126,824	0.57
256	9,355	126,926	0.76	9,669	128,968	0.74
512	9,402	129,543	0.99	9,600	130,832	0.98
Room Maps						
8	34,616	382,593	0.22	43,264	357,255	0.22
16	35,221	347,382	0.26	37,417	323,569	0.23
32	35,205	339,429	0.29	35,926	326,524	0.25
64	35,172	351,087	0.31	35,471	344,121	0.27
128	35,155	372,898	0.34	35,304	368,931	0.30
256	35,150	396,375	0.38	35,234	393,994	0.34
512	35,147	423,321	0.42	35,200	421,796	0.39

Table 3: Expansions, percolations and time per expansion in milliseconds for JPS.

Game Maps						
	h_{2^k} heuristic			Euclidean heuristic		
Neigh	Exp	Perc	Time/Exp	Exp	Perc	Time/Exp
8	45,461	4,126	0.04	43,150	4,050	0.04
16	45,198	3,964	0.06	42,054	3,855	0.06
32	44,354	3,875	0.07	41,584	3,798	0.07
64	44,194	3,859	0.08	41,803	3,804	0.08
128	44,745	3,868	0.10	42,703	3,829	0.10
256	46,305	3,903	0.13	44,562	3,875	0.13
512	49,177	3,965	0.17	47,704	3,945	0.18
Random Maps						
8	56,234	62,784	0.07	69,719	108,027	0.10
16	42,167	56,146	0.11	52,262	71,770	0.12
32	39,974	56,749	0.15	42,494	61,262	0.16
64	38,604	56,807	0.20	38,618	58,054	0.21
128	37,486	56,743	0.27	37,012	57,034	0.28
256	36,960	56,763	0.39	36,389	56,718	0.39
512	36,645	56,748	0.57	36,074	56,574	0.58
Room Maps						
8	62,924	9,381	0.05	70,599	9,872	0.05
16	74,984	8,844	0.06	75,579	8,664	0.06
32	78,460	8,580	0.06	78,264	8,454	0.07
64	80,345	8,473	0.08	79,928	8,399	0.07
128	81,117	8,463	0.09	80,634	8,419	0.09
256	81,670	8,474	0.12	81,290	8,445	0.11
512	81,817	8,484	0.15	81,518	8,463	0.15

Table 4: Expansions, percolations and time per expansion in milliseconds for Theta* and ANYA.

Game Maps			
	Exp	Perc	Time/Exp
Theta*	18,775	157,007	0.77
ANYA	8,381	78,411	0.50
Random Maps			
Theta*	7,480	93,084	0.37
ANYA	46,193	647,767	0.76
Room Maps			
Theta*	34,902	343,403	0.50
ANYA	15,145	157,316	0.48

7. Summary and Conclusions

We presented three key contributions towards the construction of effective grid path planners on the 2^k-neighborhoods. First, we formally define the 2^k-neighborhood; second, we define a 2^k distance function which can be computed in time polynomial in k and can be used as an admissible heuristic. Third, we define canonical orderings which ultimately allows us to propose a jump point search implementation of the 2^k neighborhood. Our planners produce better solutions with an increase in runtime which is smaller when using jump point search or canonical A*. Our planner produces better solutions than Theta* for the 32-neighborhood and beyond and obtains a suboptimality below 0.2% for the 64-neighborhood and beyond. Our planner, which is not particularly optimized, is faster than an implementation of the any-angle planner ANYA on average for neighborhoods with up to 64 moves, but it is not faster than the optimized algorithm iANYA. As such, our approach seems to be recommended for practitioners looking for good-quality solutions but at the same time interested in using a standard or simple implementation.

This research opens the door to future work that could investigate the use of 2^k-neighborhoods in any-angle incremental search, a problem relevant in robotics. Also relevant to robotics applications is a version of the 2^k neighborhood for 3D grids. This is one of the topics of our current research.

Acknowledgements

We thank Dietrich Daroch and Alejandro Pimentel for their comments on early drafts of this paper. Nicolás Rivera acknowledges funding from the Becas Chile initiative. Carlos Hernández and Jorge Baier are grateful to Fondecyt which partly funded this work through grants 1150328 and 1161526. Jorge Baier acknowledges funding from *Proyecto VRI Puente*.

References

Aine, S., & Likhachev, M. (2016). Truncated incremental search. *Artificial Intelligence*, *234*, 49–77.

Bailey, J., Tovey, C., Uras, T., Koenig, S., & Nash, A. (2011). Path planning on grids: The effect of vertex placement on path length. In *Proceedings of the 11th Annual International AIIDE Conference (AIIDE)*, Palo Alto, California.

Björnsson, Y., Enzenberger, M., Holte, R. C., & Schaeffer, J. (2005). Fringe Search: Beating A* at Pathfinding on Game Maps. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG)*, Essex, UK. IEEE.

Botea, A., & Harabor, D. (2013). Path planning with compressed all-pairs shortest paths data. In Borrajo, D., Kambhampati, S., Oddi, A., & Fratini, S. (Eds.), *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

Daniel, K., Nash, A., Koenig, S., & Felner, A. (2010). Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, *39*, 533–579.

Harabor, D. D., & Grastien, A. (2011). Online graph pruning for pathfinding on grid maps. In Burgard, W., & Roth, D. (Eds.), *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press.

Harabor, D. D., & Grastien, A. (2013). An optimal any-angle pathfinding algorithm. In Borrajo, D., Kambhampati, S., Oddi, A., & Fratini, S. (Eds.), *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI.

Harabor, D. D., & Grastien, A. (2014). Improving jump point search. In Chien, S. A., Do, M. B., Fern, A., & Ruml, W. (Eds.), *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI.

Harabor, D. D., Grastien, A., Öz, D., & Aksakalli, V. (2016). Optimal Any-Angle Pathfinding In Practice. *Journal of Artificial Intelligence Research*, *56*, 89–118.

Hardy, G. H., & Wright, E. M. (2008). *An Introduction to the Theory of Numbers (6th Edition)*. Oxford Mathematics, Cambridge, MA.

Hew, P. C. (2017). The length of shortest vertex paths in binary occupancy grids compared to shortest r-constrained ones. *Journal of Artificial Intelligence Research*, *59*, 543–563.

Hormazábal, N., Díaz, A., Hernández, C., & Baier, J. A. (2017). Fast and almost optimal any-angle pathfinding using the 2^{k} neighborhoods. In *Proceedings of the 10th Symposium on Combinatorial Search (SoCS)*, pp. 139–143. AAAI Press.

Kramm, B., Rivera, N., Hernández, C., & Baier, J. A. (2018). A suboptimality bound for 2^{k} grid path planning. In *Proceedings of the 11th Symposium on Combinatorial Search (SoCS)*, pp. 63–71, Stockholm, Sweden.

Lee, J., & Yu, W. (2009). A coarse-to-fine approach for fast path finding for mobile robots. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5414–5419, St. Louis, MO, USA. IEEE Press.

Marchand-Maillet, S., & Sharaiha, Y. M. (1997). Discrete convexity, straightness, and the 16-neighborhood. *Computer Vision and Image Understanding*, *66*(3), 316–329.

Nash, A. (2012). *Any-Angle Path Planning*. Doctor of Philosophy, University of Southern California.

Oh, S., & Leong, H. W. (2017). Edge n-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths. In Fukunaga, A., & Kishimoto, A. (Eds.), *Proceedings of the 10th Symposium on Combinatorial Search (SoCS)*, pp. 64–72, Pittsburgh, Pennsylvania, USA. AAAI Press.

Rabin, S., & Sturtevant, N. (2016). Combining bounding boxes and JPS to prune grid pathfinding. In *AAAI Conference on Artificial Intelligence*.

Rivera, N., Hernández, C., & Baier, J. A. (2017). Grid Pathfinding on the 2^k Neighborhoods. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, pp. 891–897, San Francisco, CA.

Rivera, N., Illanes, L., Baier, J. A., & Hernández, C. (2014). Reconnection with the ideal tree: A new approach to real-time search. *Journal of Artificial Intelligence Research*, *50*, 235–264.

Sturtevant, N. (2012). Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, *4*(2), 144 – 148.

Sturtevant, N. R., & Buro, M. (2005). Partial pathfinding using map abstraction and refinement. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pp. 1392–1397.

Sturtevant, N. R., & Rabin, S. (2016). Canonical orderings on grids. In Kambhampati, S. (Ed.), *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 683–689. IJCAI/AAAI Press.

Sturtevant, N. R., Traish, J. M., Tulip, J. R., Uras, T., Koenig, S., Strasser, B., Botea, A., Harabor, D., & Rabin, S. (2015). The grid-based path planning competition: 2014 entries and results. In Lelis, L., & Stern, R. (Eds.), *Proceedings of the 8th Symposium on Combinatorial Search (SoCS)*, p. 241. AAAI Press.

Tsitsiklis, J. N. (1995). Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, *40*(9), 1528–1538.

Uras, T., & Koenig, S. (2015). An empirical comparison of any-angle path-planning algorithms. In Lelis, L., & Stern, R. (Eds.), *Proceedings of the 8th Symposium on Combinatorial Search (SoCS)*, pp. 206–211. AAAI Press. Code available at: `http://idm-lab.org/anyangle`.

Uras, T., Koenig, S., & Hernández, C. (2013). Subgoal graphs for optimal pathfinding in eight-neighbor grids. In Borrajo, D., Kambhampati, S., Oddi, A., & Fratini, S. (Eds.), *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI.