Fast and Almost Optimal Any-Angle Pathfinding Using the 2^k Neighborhoods

Nicolás Hormazábal, Antonio Díaz, Carlos Hernández
Departamento de Ciencias de la Ingeniería
Universidad Andrés Bello
Santiago, Chile

Jorge A. Baier
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Any-angle path finding on grids is an important problem with applications in autonomous robot navigation. In this paper, we show that a well-known pre-processing technique, namely subgoal graphs, originally proposed for (non any-angle) 8-connected grids, can be straightforwardly adapted to the 2^k neighborhoods, a family of neighborhoods that allow an increasing number of movements (and angles) as k is increased. This observation yields a pathfinder that computes 2^k-optimal paths very quickly. Compared to ANYA, an optimal true any-angle planner, over a variety of benchmarks, our planner is one order of magnitude faster while being less than 0.0005% suboptimal. Important to our planner's performance was the development of an iterative 2^k heuristic, linear in k, which is also a contribution of this paper.

Introduction

Optimal any-angle grid pathfinding is the problem of finding a shortest path between two points on a grid, allowing any straight movement between two grid points that does not intersect an obstacle. It is an important problem with applications in autonomous robot navigation over 2D terrain.

Although planners specifically designed to generate any-angle paths do exist—two examples are Theta* (Daniel et al. 2010) and ANYA (Harabor et al. 2016)—, the question of whether or not similar or better performance can be achieved using standard grid pathfinding with bounded connectivity seems still open. This is particularly intriguing given the recent development of the 2^k neighborhood (Rivera, Hernández, and Baier 2017), a family of neighborhoods in which 2^k movements are allowed in each cell. As k is increased, more movements and thus more angles are available in each cell.

The motivation for the work we report in this paper was to investigate the result of applying 2^k neighborhoods to one of the fastest, optimal approaches to pathfinding in 8-connected grids: subgoal graphs (Uras, Koenig, and Hernández 2013). We chose this technique because of its key role in planners that have obtained remarkable performance in the Grid-Based Path Planning Competition (GPPC) (Sturtevant et al. 2015). Subgoal graphs is a preprocessing technique; that is, it is only applicable when the map is known in advance.

In this paper, we show that it is straightforward to adapt subgoal graph preprocessing to the 2^k-neighborhoods. While Uras, Koenig, and Hernández (2013) rely strongly on the *octile distance* to compute their graph, our adaptation relies on the h_{2^k} heuristic of Rivera, Hernández, and Baier (2017). To improve performance of the resulting system, we develop a novel iterative algorithm for computing h_{2^k}, which is simple to implement and linear in k, unlike heuristics previously described.

We compared our implementation to subgoal-graph-based planners and to ANYA, an optimal any-angle pathfinder. We show our system computes nearly any-angle optimal paths (indeed, 0.0003%-suboptimal). Our planner outperforms Sub2(Theta*) (Uras and Koenig 2015a), the current best existing system for suboptimal any-angle planning based on the subgoal graph preprocessing. Moreover, it is over one order of magnitude faster than ANYA.

Background

Grid Pathfinding

An $N \times M$ grid is the set of ordered pairs $G = \{(i, j) \mid 0 \leq i \leq N, 0 \leq j \leq M\}$. Traditionally, in grid pathfinding an agent is assumed to be located at the center of the cells of the grid. Here we follow Uras and Koenig (2015b) and assume the agent moves between the vertices of cells.

At each cell (a, b) the agent is allowed a number of moves, which are taken from *neighborhood*, $\mathcal{N}$. Moves are ordered pairs of integers. Below we treat ordered pairs as vectors and use boldface to denote them. We say (a, b) is clockwise-left of (c, d), denoted $(a, b) \preceq (c, d)$, iff $ad \leq cb$. We define $\|(x, y)\|$ as $\sqrt{x^2 + y^2}$.

A $N \times M$ *map* is a tuple (G, O) where G is an $N \times M$ grid and $O \subseteq G$ is a set of *obstacle cells*. Intuitively, when $\mathbf{o}$ is an obstacle, the cell whose lower-left corner is $\mathbf{o}$ is not traversable. A move $\mathbf{m} \in \mathcal{N}$ is *legal* in cell $\mathbf{c}$ iff (1) $\mathbf{c}+\mathbf{m} \in G$, and (2) when the segment between $\mathbf{c}$ and $\mathbf{c} + \mathbf{m}$ does not penetrate any of the obstacle cells. Intuitively, a move penetrates an obstacle $\mathbf{o} \in O$ when it penetrates the square cell associated with $\mathbf{o}$. Formally, a movement $\mathbf{m}$ on cell $\mathbf{c}$ penetrates an obstacle cell $\mathbf{o}$ iff there exist $\lambda \in [0, 1]$ and $\mu, \nu \in (0, 1)$ such that the following equation holds:

$$\mathbf{c} + \lambda\mathbf{m} = \mathbf{o} + \mu(1, 0) + \nu(0, 1).$$

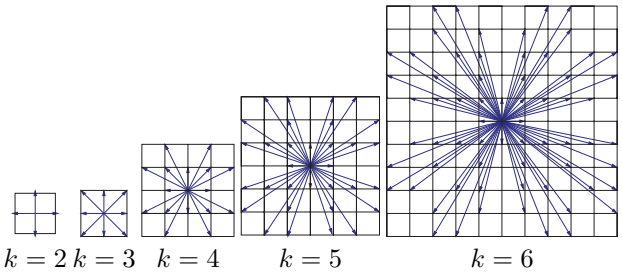

Figure 1: The 4-, 8-, 16-, 32-, and 64- neighborhoods.

The set of successors of a cell $\mathbf{u}$ is defined as $Succ_{\mathcal{N}}(\mathbf{u}) = \{\mathbf{m}+\mathbf{u} \mid \mathbf{m} \in \mathcal{N} \text{ and } \mathbf{m} \text{ is legal in } \mathbf{u}\}$. A *path over* $\mathcal{N}$ from $\mathbf{u}$ to $\mathbf{v}$ is a sequence of cells $\mathbf{c}_1\mathbf{c}_2 \cdots \mathbf{c}_n$ such that $\mathbf{c}_1 = \mathbf{u}$, $\mathbf{c}_n = \mathbf{v}$, and for every $i \in \{1, \ldots, n-1\}$ it holds that $\mathbf{c}_{i+1} \in Succ_{\mathcal{N}}(\mathbf{c}_i)$. The cost of a path $\sigma = \mathbf{c}_1 \cdots \mathbf{c}_n$ is $c(\sigma) = \sum_{i=1}^{n-1} \|\mathbf{c}_{i+1} - \mathbf{c}_i\|$. A path over $\mathcal{N}$ σ from $\mathbf{u}$ to $\mathbf{v}$ is *optimal* if for every path σ' over $\mathcal{N}$ from $\mathbf{u}$ to $\mathbf{v}$ it holds that $c(\sigma) \leq c(\sigma')$. A path σ *traverses* a cell $\mathbf{c}$ iff $\mathbf{c}$ is in σ.

A grid pathfinding problem is a tuple $P = (G, O, \mathcal{N}, \mathbf{u}_{start}, \mathbf{u}_{goal})$, where (G, O) is a map, $\mathcal{N}$ is a neighborhood, $\mathbf{u}_{start} \in G$ is the initial cell, and $\mathbf{u}_{goal} \in G$ is the goal cell. 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 $\mathcal{N}$.

The 8-connected (octile) neighborhood has been used traditionally to evaluate grid pathfinding algorithms. It is defined as $\mathcal{N}_8 = \{(i, j) \mid i, j \in \{-1, 0, 1\}, |i| + |j| > 0\}$. 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. We denote the octile between $\mathbf{u}$ and $\mathbf{v}$ as $h_8(\mathbf{u}, \mathbf{v})$.

2^k Neighborhoods Rivera, Hernández, and Baier (2017), generalize the octile neighborhood to 2^k-connected neighborhoods, for any natural $k \geq 2$. The 2^k-neighborhood, which we denote by $\mathcal{N}_{2^k}$, is in turn defined using a series $\mathcal{Q}_0, \mathcal{Q}_1, \ldots$, where $\mathcal{Q}_m$, for any natural m, contains the first-quadrant moves of the $\mathcal{N}_{2^{m+2}}$ neighborhood. Assuming that given a sequence, $\sigma = a_0 \cdot a_1 \cdots a_n$, σ^j denotes the element a_j, sequence $\{\mathcal{Q}\}_i$ is inductively defined as follows: $\mathcal{Q}_0 = (1, 0) \cdot (0, 1)$ and the j-th element in $\mathcal{Q}_m$, for every $m > 0$, is given by:

$$\mathcal{Q}_m^j = \begin{cases} \mathcal{Q}_{m-1}^{j/2}, & \text{if } j \text{ is even,} \\ \mathcal{Q}_{m-1}^{\lfloor j/2 \rfloor} + \mathcal{Q}_{m-1}^{\lceil j/2 \rceil} & \text{otherwise,} \end{cases} \quad (1)$$

where j is an integer in $\{0, \ldots, 2^m + 1\}$. Finally, $\mathcal{N}_{2^k} = \{(\pm x, \pm y) \mid (x, y) \in \mathcal{Q}_{k-2}\}$. Figure 1 depicts $\mathcal{N}_{2^k}$ as k increases.

The Any-Angle Neighborhood Any-Angle grid pathfinding is obtained when using a neighborhood that allows reaching any point in the grid. Formally, the any-angle neighborhood for a grid G is defined as $\mathcal{N}_{any}^G = \{(\pm x, \pm y) \mid (x, y) \in G)\}$.

Subgoal Graphs

Given a problem $P = (G, O, \mathcal{N}, \mathbf{u}_{start}, \mathbf{u}_{goal})$, one can use an admissible heuristic along with A* to find optimal solutions. Nevertheless, if the map and neighborhood are known in advance, there are preprocessing techniques that can be applied to speed up search. A well-known preprocessing technique for 8-connected maps is subgoal graphs (Uras, Koenig, and Hernández 2012).

Building upon the fact that optimal solutions always "touch the corners of obstacles", a *subgoal graph*, where the nodes of the graph are subgoals, is built in the following way. Given an obstacle $o = (i, j)$, we define the *corners of* o as the set $\{(i, j), (i+1, j), (i, j+1), (i+1, j+1)\}$. Now, given a map (G, O), $\mathbf{s} \in G$ is a subgoal iff (1) s is the corner of an obstacle $o \in O$ and (2) both $\mathbf{d}$ and $-\mathbf{d}$ are legal moves in $\mathbf{s}$, for some $\mathbf{d} \in \{(1, 1), (-1, 1)\}$. To define the connectivity of the subgoal graph Uras, Koenig, and Hernández (2012) introduce the following definitions.

Definition 1 (h_8-reachability) *Given a map (G, O), two subgoals $\mathbf{s}_1$ and $\mathbf{s}_2$ are h_8-reachable iff $h_8(\mathbf{s}_1, \mathbf{s}_2)$ is the cost of an optimal path over $\mathcal{N}_8$ between $\mathbf{s}_1$ and $\mathbf{s}_2$.*

We note that Uras, Koenig, and Hernández use the term "h-reachable" since they only deal with one neighborhood (the octile neighborhood). In this paper we slightly change the terminology for accuracy.

Definition 2 (direct h_8-reachability) *Given a map (G, O) and a set S of subgoals, two h_8-reachable subgoals $\mathbf{s}_1$ and $\mathbf{s}_2$ ($\mathbf{s}_1, \mathbf{s}_2 \in S$) are* directly h_8-reachable *if no subgoal $\mathbf{s}_3 \in S$ is traversed by any optimal path over $\mathcal{N}_8$ from $\mathbf{s}_1$ to $\mathbf{s}_2$.*

For a map $M = (G, O)$ a subgoal graph $\mathcal{S} = (S, E)$ is such that S is the set of subgoals over M and $E = \{(\mathbf{s}_1, \mathbf{s}_2) \mid \mathbf{s}_1 \text{ and } \mathbf{s}_2 \text{ are directly } h_8\text{-reachable}\}$. The cost of each arc is $w(\mathbf{s}_1, \mathbf{s}_2) = h_8(\mathbf{s}_1, \mathbf{s}_2)$.

A *2-level subgoal graph* is built from a subgoal graph (S, E, w) as follows. We identify a maximal subset S_2 of subgoals in S such that the cost of the optimal path between any pair $\mathbf{s}_1, \mathbf{s}_2$ of subgoals in $S \setminus S_2$ over the graph whose only nodes are $S \setminus S_2$ is equal to the cost of the optimal path from $\mathbf{s}_1$ to $\mathbf{s}_2$ over graph (S, E). The 2-level subgoal graph is defined as the graph induced in (S, E) by $S \setminus S_2$.

To find an optimal path over $\mathcal{N}_8$ from $\mathbf{u}$ to $\mathbf{v}$ over a map (G, O) we (1) compute the 2-level subgoal graph $\mathcal{S}$ (2) compute the subgoals that are directly h_8-reachable from $\mathbf{u}$ and from $\mathbf{v}$ (3) build a new graph $\mathcal{S}_{\mathbf{u},\mathbf{v}}$ containing the nodes and arcs in S adding the arcs computed previously, with the appropriate costs, and (4) run A* from $\mathbf{u}$ to $\mathbf{v}$ in $\mathcal{S}_{\mathbf{u},\mathbf{v}}$. The correctness of this is justified by the following result.

Theorem 1 *Let $P = (G, O, \mathcal{N}, \mathbf{u}_{start}, \mathbf{u}_{goal})$ and $\mathcal{S}$ the 2-level graph extended with $\mathbf{u}_{start}$ and $\mathbf{u}_{goal}$. There exists a linear function that takes as input a shortest path over $\mathcal{S}$ and returns an optimal solution for P.*

An Iterative 2^k Heuristic

Rivera, Hernández, and Baier (2017) describe a simple two-step method for obtaining an admissible heuristic for the $\mathcal{N}_{2^k}$ neighborhood. Specifically, to compute the distance between $(0,0)$ and a first-quadrant position (x,y), in the first step one has to find the two consecutive movements in $\mathcal{Q}_{k-2}$, say $\mathbf{u}$ and $\mathbf{v}$, such that $\mathbf{u} \preceq (x,y) \preceq \mathbf{v}$. For this first step, they note that finding such $\mathbf{u}$ and $\mathbf{v}$ can be carried out using a linear search over the movements in $\mathcal{Q}_{k-2}$, which is exponential in k. Alternatively, they mention that a binary search scheme, which is not described in detail, can also be used. As a second step, they solve the system of two equations for variables p and q yielded by:

$$p\mathbf{u} + q\mathbf{v} = (x,y). \tag{2}$$

The output distance between $(0,0)$ and (x,y) is $p\|\mathbf{u}\| + q\|\mathbf{v}\|$. In their experimental evaluation, they use functions that do not use binary search. Indeed, for $k=5$, they present the function shown in Algorithm 1, which essentially carries out a linear search over the movements in $\mathcal{Q}_3$.

Algorithm 1: A distance for the 32-connected neighborhood.

```
function h32(x, y)
    if x > y then swap x and y
    if 3x < y then return (y − 3x) + √10 x
    else if 2x < y then return √10(y − 2x) + √5(3x − y)
    else if 3x < 2y then return √5(2y − 3x) + √13(2x − y)
    else return √13(y − x) + √2(3x − 2y)
```

There are disadvantages associated with using functions such as that of Algorithm 1. Given that the heuristic function is computed many times during an A* search, we need efficient implementations. But a linear-search scheme like that of Algorithm 1 does not scale with k. Second, an obvious way of transforming Algorithm 1 into a binary-search scheme would be to re-order the if statements such that the depth of the checks is lower (logarithmic in 2^k). But such a re-ordering would result in a piece of code whose size is still exponential in k. Indeed, the size of such a function would be similar to that of Algorithm 1. Third, we observe that for every k we obtain a *different* heuristic function, which of course requires a different implementation.

We propose an algorithm that addresses all three of the disadvantages we discussed above. The key observation is that to compute the heuristic it is not necessary to carry out a two-step approach. Our function receives k as a parameter, has constant size, and does not need to generate $\mathcal{Q}_{k-2}$ in memory. At an abstract level, it 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 a 2^k neighborhood. To illustrate this, imagine we want 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

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

```
function distance(x, y, k)
    l ← (1, 0)
    r ← (0, 1)
    while k > 2 do
        if x > y then
            r ← r + l
            x ← x − y
        else
            l ← r + l
            y ← y − x
        k ← k − 1
    return x‖r‖ + y‖l‖
```

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 k.

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

Proof: Observe that with each iteration, we introduce a movement of the next neighborhood, thus at end of the n-th iteration $\mathbf{l}$ and $\mathbf{r}$ are always consecutive movements of $\mathcal{Q}_n$; this follows from the definition of $\{Q\}_j$. Second, observe that $(a,b) = x\mathbf{l} + y\mathbf{r}$ is an invariant of the loop. Finally, observe $\mathbf{l} \preceq (a,b) \preceq \mathbf{r}$ follows from the expression of factorization and the the fact that x and y are non-negative. Using the optimality theorem of Rivera, Hernández, and Baier (2017), we conclude the returned value is the distance between $(0,0)$ and (a,b) on $\mathcal{N}_{2^k}$. ■

We denote cost of an optimal path between two cells $\mathbf{u}$ and $\mathbf{v}$ over an obstacle-free grid with the $\mathcal{N}_{2^k}$ as $h_{2^k}(\mathbf{u},\mathbf{v})$.

Iterative h_{2^k} in Practice We evaluated the performance of iterative h_{2^k} versus the original h_{2^k}, for $k = 5,6,7$. We found that our iterative version requires between 65% and 75% of the time required by original h_{2^k}.

Subgoal Graphs with the 2^k Neighborhood

It is straightforward to extend the notion of subgoal graph for $\mathcal{N}_{2^k}$. First we do not need to change the defintion for subgoal. Second, we define h_{2^k}- and direct h_{2^k}- reachability by replacing h_8 and $\mathcal{N}_8$, respectively, by h_{2^k} and $\mathcal{N}_{2^k}$, on Definitions 1 and 2. To define 2-level 2^k subgoal graphs, we

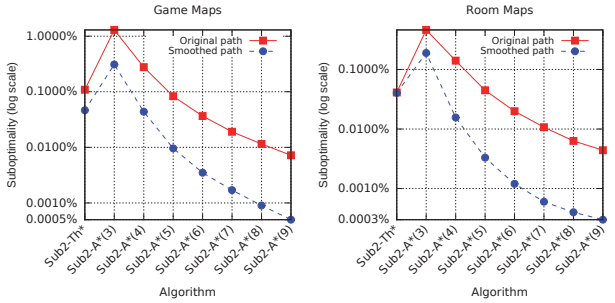

Figure 2: Suboptimality Percentage Ratio (log-scale on the Y axis).

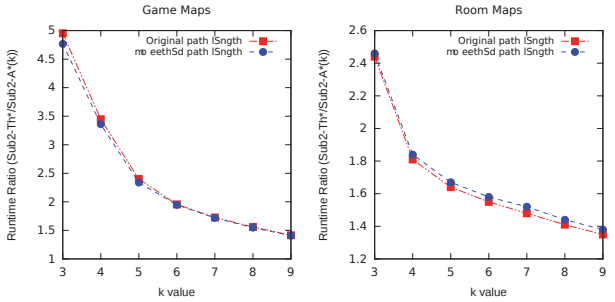

Figure 3: Runtime Ratio.

simply use h_{2^k} instead of h_8 in the method described earlier. Finally, an analogous version of Theorem 1 is straightforward to prove for h_{2^k}, allowing us to use A* over the subgoal graph to compute optimal paths on the original map.

From a computational perspective, we compute 2-level h_{2^k} subgoal graphs using the same algorithms proposed by Uras, Koenig, and Hernández (2012). In practice, an h_{2^k} subgoal graph may have more connections than an h_8 subgoal graph because pairs of subgoals that used to be h_8-reachable (hence h_{2^k}-reachable) but not directly h_8-reachable now indeed could also be directly h_{2^k}-reachable, since $\mathcal{N}_{2^k}$ allows more paths between cells.

Empirical Evaluation

We implemented our algorithm, Sub2-A*(k), on top of Uras and Koenig's Sub2 implementation (2015b). In addition to building subgoals, our planner implements the post-processing smoothing method described by Botea, Müller, and Schaeffer (2004), a simple linear-time algorithm that replaces consecutive moves by a longer move if the two endpoints are line-of-sight. We compare to ANYA, an optimal any-angle planner, and to Sub2-Theta* (Uras and Koenig 2015b). Sub2-Theta* can be regarded as the best existing any-angle planner that uses the subgoal graph technique for preprocessing. In their evaluation, this algorithm shows a good performance and balance between suboptimality and runtime. Sub2-Theta* builds a subgoal graph and searches over it using Theta*.

We use three sets of game maps in our comparison, namely, maps from the game Baldurs Gate II of size 512×512, maps from the game Dragon Age: Origins of Size ranging from 22×28 to 1260×1104, and maps from the game StarCraft ranging from 384×384 to 1024×1024. Additionally, we use random maps of size 512×512, where the percentage of blocked cells varies from 10 percent to 40 percent, and room maps of size 512×512 where the size of the room vary from 8×8 to 64×64. We evaluated all the instances provided in Nathan Stutervant's repository (Sturtevant 2012). The experiments on a 2.20GHz Intel(R) Xeon(R) CPU machine with 128GB of RAM.

As a measure of solution quality, we use the *suboptimality percentage*, defined as 100 times the difference between the cost of the solution found and the cost of the optimal any-angle solution divided by the cost of the optimal any-angle solution. As a measure of efficiency, we use runtime. For space, we only show plots for the Game and Room Maps, and omit Random Maps. Those plots, however, look similar to the other two benchmarks.

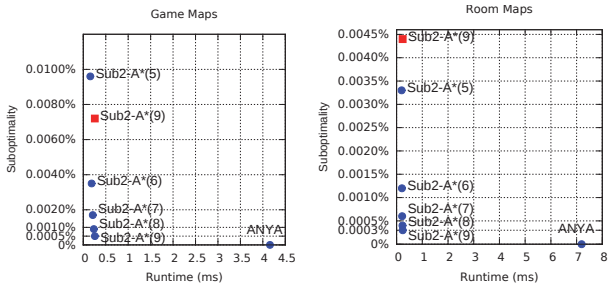

Figure 4: Sub2-A*(k) vs. ANYA

Comparing Sub2-Theta* with Sub2-A*(k)

Figure 2 shows the suboptimality percentage obtained by Sub2-Theta* and that obtained by Sub2-A*(k), for different values for k. A number n larger than 1 means that Sub2-A*(k) is n times less suboptimal than Sub2-Theta*. When $k > 5$, Sub2-A*(k) can be orders of magnitude less suboptimal than Sub2-Theta* when smoothing is applied. Figure 3 shows the ratio between the runtime obtained by Sub2-Theta* and the runtime obtained by Sub2-A*(k). Sub2 A*(k) is faster than Sub2-Theta* for all values of k we used. Note that when the value of k increases the runtime of Sub2-A*(k) increases too. The main reason is that the branching factor of the subgoal graphs increases with k. For instance, in room maps the average branching factor increases from 5.3 ($k = 3$) to 9.3 ($k = 9$), and in Baldurs Gate II it increases from 9.3 ($k = 3$) to 19.8 ($k = 9$).

Comparing ANYA with Sub2-A*(k)

Figure 4 shows suboptimality percentage versus runtime for ANYA and Sub2-A*(k) for some values of k. We use blue circles for smoothed paths and red squares for non-smoothed paths. We did not include Sub2-Theta* and other (small) values of k since they were off the chart (substantially worse than the other algorithms shown). We observe that Sub2-A*(k) can be orders of magnitude faster than ANYA; on the other side Sub2-A*(k) can obtain a suboptimality very close to 0 percent.

We compared to other any-angle algorithms that appear in Uras and Koenig's paper (2015a) with Sub2-A*(k). Sub2-A*(k) has superior performance regarding suboptimality and runtime than all other algorithms. We do not include the results for lack of space.

Summary

We presented Sub2-A*(k), an almost-any-angle planner that computes almost optimal any-angle paths for grid pathfinding. Our planner is obtained by applying the recently developed 2^k neighborhoods on top of subgoal graph preprocessing. In addition, we presented a novel general algorithm, linear in k, for computing the h_{2^k} heuristic. This algorithm is used both during preprocessing and during search.

In our experimental evaluation, we show that our planner substantially outperforms existing any-angle planners based on the subgoal graph approach. Moreover, it outperforms the optimal ANYA planner in terms of runtime by an order of magnitude, while being only 0.0005% suboptimal.

Acknowledgements

We acknowledge support from Fondecyt via grants number 1150328 and number 1161526.

References

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Using component abstraction for automatic generation of macro-actions. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, 181–190. AAAI.

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

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

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

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

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

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

Uras, T., and Koenig, S. 2015b. Speeding-up any-angle path-planning on grids. In Brafman, R. I.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 234–238. Jerusalem, Israel: AAAI Press.

Uras, T.; Koenig, S.; and Hernández, C. 2012. Subgoal graphs for eight-neighbor gridworlds. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *Proceedings of the 5th Symposium on Combinatorial Search (SoCS)*. AAAI Press.

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