Path Planning with CPD Heuristics

Massimo Bono[1] , **Alfonso E. Gerevini**[1] , **Daniel D. Harabor**[2] and **Peter J. Stuckey**[2]
[1]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Brescia, Italy
[2]Faculty of Information Technology, Monash University, Melbourne, Australia
{mbono, alfonso.gerevini}@unibs.it, {daniel.harabor, peter.stuckey}@monash.edu

Abstract

Compressed Path Databases (CPDs) are a leading technique for optimal pathfinding in graphs with static edge costs. In this work we investigate CPDs as admissible heuristic functions and we apply them in two distinct settings: problems where the graph is subject to dynamically changing costs, and anytime settings where deliberation time is limited. Conventional heuristics derive cost-to-go estimates by reasoning about a tentative and usually infeasible path, from the current node to the target. CPD-based heuristics derive cost-to-go estimates by computing a concrete and usually feasible path. We exploit such paths to bound the optimal solution, not just from below but also from above. We demonstrate the benefit of this approach in a range of experiments on standard gridmaps and in comparison to Landmarks, a popular alternative also developed for searching in explicit state-spaces.

1 Introduction

There has been massive progress in quickly computing shortest paths for graphs with static edge-costs. Modern algorithms such as Contraction Hierarchies [Geisberger *et al.*, 2008], Hub Labels [Abraham *et al.*, 2012] and Compressed Path Databases [Botea, 2011; Botea and Harabor, 2013] use a combination of auxiliary data structures and fast query solving to improve the performance of reference algorithms such as Dijkstra search by several orders of magnitude. When edge costs are dynamic however fast path planning algorithms no longer guarantee optimality, and may fail altogether, due to invalid auxiliary data. When this happens the auxiliary data must be repaired or recomputed entirely from scratch. Problems with dynamic edge-costs are therefore more challenging than the static-cost case, and they appear in a number of important settings; e.g.:

- Routing in road networks, where travel times are adjusted due to congestion, construction and road closures [Delling and Wagner, 2007];
- Personalised routing, where edges have agent-dependent costs [Delling *et al.*, 2011; Dibbelt *et al.*, 2014; Funke and Storandt, 2015].
- Computer video games, where player interaction creates changes to the environment [Kring *et al.*, 2010].

In this work we consider how to solve dynamic-cost path planning problems using Compressed Path Databases (CPDs): a preprocessing-intensive technique that finds static shortest paths without any state-space search. In dynamic-cost settings CPD auxiliary data is invalidated and computed paths are no longer optimal. However, CPDs can still be used as heuristic functions to drive A* search and here they have several unique advantages. For example, each time a node is expanded we always have available a concrete and usually feasible plan to reach the target (i.e. the CPD path). Moreover, the static and dynamic costs of each CPD path provide strong upper and lower bounds that help A* to find the true optimal path sooner. Finally, when the upper and lower bounds coincide, A* can be immediately terminated, often well before it is necessary to expand the target.

We explore two possible applications of CPD heuristics. The first is optimal grid search, where we report convincing gains of up to several factors vs Landmarks [Goldberg and Harrelson, 2005], a popular but conventional preprocessing-based heuristic. We also consider CPD heuristics in Anytime A* and Anytime Weighted A* [Hansen and Zhou, 2007], a popular algorithm for settings where deliberation time is limited. Here we demonstrate even larger benefits.

2 Problem Setting

We study path planning problems in weighted graphs with dynamically changing edge costs. For input we require a graph $G = (N, E)$ with nodes N and edges $E \subseteq N \times N$. Associated with the graph is a non-negative function $w : N \times N \rightarrow \mathbb{R}^+$, where $w(m, n)$ is the weight of edge $(m, n) \in E$.

Each *instance* of the path planning problem is defined by a start node s and a target node t. Our task is to find a *path* in G from s to t where a path P is defined as a sequence of edges $P = [(s = n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k = t)]$. The *cost* of the path P given w, denoted $c(P, w)$, is exactly:

$$c(P, w) = \sum_{i=1}^{k-1} w(n_i, n_{i+1})$$

We say that P is *optimal* if $c(P, w) \leq c(P', w)$ among all paths P' from s to t. We use [] to denote the empty sequence and $+\!\!+$ to denote sequence concatenation.

Context. In this work we focus on the path planning problem. That means we assume the start and target locations are fixed for the duration of each path planning episode (i.e. while solving a single instance) and we assume that any two subsequent path planning episodes are unrelated. We refer to the period of time, from the beginning of the first path planning episode to the end of the last episode, as the *online phase*. Consistent with typical road routing and computer game settings, we also assume that the graph is known a priori and that that it can be preprocessed during an *offline phase*. That means any offline investments in time, such as might be required to construct auxiliary data, do not need to be amortised online (i.e., the two phases are entirely divorced). From time to time during the online phase we allow edge costs to change as a result of some unknown exogenous event. We refer to such changes as *perturbations* and we assume they occur only between distinct path planning episodes.[1] The cost increases are always positive with respect to the original edge value, as observed at the time the graph is loaded. A related but different problem to ours, also involving dynamically changing edge costs, appears in the context of *incremental search*. Here planning, execution and sensing are interleaved and the map can change while an agent executes a path. Such settings are beyond our scope and we refer the interested reader to [Koenig *et al.*, 2004] for an overview.

3 Related Work

Perhaps most similar to our work is ALT [Goldberg and Harrelson, 2005], a popular search technique that uses precomputed *landmark heuristics* to guide search in static and dynamic-cost graphs. ALT stores a precomputed set of distance labels: for every node in the graph to and from a selected set of *landmark* nodes. These distances are exploited to derive admissible lower-bounds on the true distance between any pair of graph nodes. For example, let L be the set of landmark nodes and a and b any pair of nodes in the graph. Then, $h(a, b) = \arg\max\{\ |d(a, l) - d(b, l)|\ \}\ s.t.\ l \in L$.

Several variations of this idea appear in the literature with a summary and unifying treatment given in [Sturtevant *et al.*, 2009]. One attractive property of landmarks is that the estimates are robust [Delling and Wagner, 2007] and remain admissible as long as edge-costs never drop below their initial (i.e. unperturbed) values. More landmarks produce more accurate estimates but since each landmark requires $2 \times |N|$ distance labels the memory overhead grows quickly prohibitive. A related issue is the growing online cost to evaluate many landmarks and to find their maximum. For these reasons the number of landmarks is usually limited to a small set.

A variety of other methods exists in the literature, originally developed for static routing in road networks, but which can be adapted to the dynamic case. Contraction Hierarchies (CH) [Geisberger *et al.*, 2008] is a fast, optimal and broadly representative example; another is Dynamic Highway Node Routing [Schultes and Sanders, 2007]. In the event of edge-cost changes CH auxiliary data can be repaired [Geisberger *et al.*, 2012] but the cost of this operation can dominate running time, unless updates are small or infrequent.

[1] Equivalently, we consider only periodic (cf. real-time) updates.

W	W	W	W,E	E	E	E
W	W	W	W,E	E	E	E
W	W				E	E
W	W	W	*s*	E	E	E
W,SW	W,SW	SW	S	SE	E,SE	E,SE

Figure 1: We show all optimal first move(s): from the indicated (yellow) source node s, to all other nodes.

Another way to improve on reference algorithms in dynamic road networks is to compute a *metric independent* auxiliary data structure and then *customise* this data with a user-specified metric function. Two examples of this idea are CCH [Dibbelt *et al.*, 2014] and CRP [Delling *et al.*, 2011]. The main drawback is the customisation overhead. Even a highly engineered implementation running on 12 parallel cores [Delling *et al.*, 2017] requires ≈ 1 second per customisation plus the cost of computing a shortest distance and extracting a corresponding shortest path; i.e. depending on the problem, and the frequency of customisation calls, it could well be faster to simply run the reference algorithm. Two recent and related methods, PCH and PRP [Funke and Storandt, 2015], can significantly improve customisation performance but they require that metrics be available in advance.

4 Compressed Path Databases

We now review some key aspects of CPDs. Our descriptions are based on SRC [Strasser *et al.*, 2015], a leading variant in this algorithmic family. In simple words, a CPD is a data structure that stores optimal moves: from any node s on a graph towards any other node t. More formally, a "move" from s towards t is the first edge of a path from s to t. An *optimal move* is the first move of an optimal path.[2]

Building a CPD. It is an offline procedure that requires a number of iterations, one for each node in the graph. Each iteration is a complete Dijkstra search from a given source node s. With only slight modification, the Dijkstra algorithm produces a so-called *first-move* table $T(s)$. In a first-move table (see Figure 1 for an example) all graph nodes t are assigned a label that identifies which of the edges leaving s appear on a shortest path towards t. The first-move table is then compressed, which concludes the iteration at hand. Being independent each Dijkstra search can be run in parallel, with a speed-up linear in the number of available processors.

Compression. It reduces the size of a first-move table using run-length encoding. For example, suppose the nodes in Figure 1 are ordered from left to right and from the top to the bottom. A string of symbols such as W; W; W; (W, E) can then be more compactly represented as 1W (i.e., a solid block of Ws starting at position 1). Observe how we are free to choose any symbol from a non-singleton list such as (W,E). Furthermore, obstacle nodes and the origin node s can be treated as

[2] We use the terms "move" and "edge" interchangeably.

wildcards, i.e. compatible with any run [Salvetti *et al.*, 2017]. The effectiveness of RLE compression depends strongly on the the way nodes are ordered (i.e. we want to choose a *good* ordering that produces few runs across all source nodes). The problem is intractable in general but good heuristics exist. We use the DFS ordering from [Strasser *et al.*, 2015].

Path Extraction. It is an online procedure which depends on CPDs to find optimal shortest paths. We denote as $\mathsf{CPD}(s,t)$ a function which returns the first move from s on an optimal shortest path to t. This function requires only a simple binary search through the compressed string of symbols for s. Once the first move is known we apply it directly and call the function again, recursively and until the target is reached.

5 Searching with CPD Heuristics

The principal idea we explore in this paper is the use of CPD as a lower bound heuristic for path planning. Under the perturbation scenario we consider, the cost of the shortest path recorded in the CPD using original weights w is a lower bound on the actual shortest path in the graph using new weights w'. Clearly since $w'(s,t) \geq w(s,t)$ for all $(s,t) \in E$ we have that $c(P,w') \geq c(P,w)$ for all paths P in G. Hence the CPD defines an *admissible heuristic* for A* search.

Furthermore, since the path P recorded in the CPD can be quickly recovered, we can calculate its cost $c(P,w')$ using the new weights w'. This allows us to track an incumbent solution which is the shortest such path we have found. Given that we have an upper bound on the solution, and the f costs of nodes represent a lower bound of the solution we can then adjust the A* algorithm to return a *bounded suboptimal path* whose cost is not more than ϵ times the shortest path, where $\epsilon \geq 1$. This may allow us to explore much less of the graph than if we demand to find an optimal path.

Algorithm 1 is a modification of A* as follows. For simplicity, we store with each node n in the open list a (current) shortest path $p[n]$ from s to n; in practice this is stored by back pointers. We keep track of a shortest incumbent path I and its cost u (an upper bound on the cost of the shortest possible path). The incumbent I is stored as a node m which encodes the concatenation of the current path from s to m with the CPD path from m to t.

When we select a node n from the *open* set (line 6), if it is the target we have found an optimal path and we return it (line 7). If ϵ times the f value is not less than u, then the incumbent path I is no worse than factor ϵ from optimal, and we return the incumbent (line 8) (function $\mathsf{SP\text{-}CPD}(I,t)$ returns the path recorded by the CPD from I to t). Note that the algorithm will always *terminate early* if it expands a node n, where $\mathsf{wCPD}(n,t)$ indicates that $h = h'$. In this case, when n was added to *open* it also became the incumbent I and set $u = f[n] = g[n] + h$, and hence the test on line 8 succeeds.

Otherwise we investigate the neighbours m of n where the path from s to m via n is shorter than any previously found path. We call $\mathsf{wCPD}(m,t)$ which returns the cost $h = c(P,w)$ according to w of the path P recorded in the CPD from m to t, as well as the cost $h' = c(P,w')$ according to w'. We update the f cost of m using the original cost h of this path (line 16).

Algorithm 1: $\mathsf{CPD\text{-}Search}(w, w', s, t, \epsilon)$: Variant of Weighted A* with a CPD heuristic. Parameters s and t indicate the start and target, w' encodes actual edge-costs while w encodes weights used by the CPD. This algorithm guarantees solutions are ϵ-optimal.

```
1  closed ← ∅; open ← {s}
2  for n ∈ N do g[n] ← ∞;
3  g[s] ← 0; f[s] ← 0; p[s] ← []
4  u ← ∞; I ← s
5  while open ≠ ∅ do
6      n ← arg min{f[n'] | n' ∈ open}
7      if n = t then return p[n] ;
8      if ε f[n] ≥ u then return p[I] ++SP-CPD(I, t) ;
9      open ← open − {n}
10     closed ← closed ∪ {n}
11     for (n, m) ∈ E, m ∉ closed do
12         if g[n] + w'(n, m) < g[m] then
13             p[m] ← p[n] ++ [(n, m)]
14             g[m] ← g[n] + w'(n, m)
15             ⟨h, h'⟩ ← wCPD(m, t)
16             f[m] ← g[m] + h
17             if u > g[m] + h' then
18                 u ← g[m] + h'; I ← m
19             open ← open ∪ {m}
20 return ⊥                                  ▶ No solution
```

Algorithm 2: $\mathsf{wCPD}(s,t)$: Retrieving the original (using w) and new (using w') cost of the shortest path (according to w) for an (s,t) pair.

```
1 if s = t then return (0,0);
2 if co[s] < ∞ then return ⟨co[s], cn[s]⟩;
3 (s, m) ← CPD(s, t)
4 ⟨h, h'⟩ ← wCPD(m, t)
5 co[s] ← h + w(s, m)
6 cn[s] ← h' + w'(s, m)
7 return ⟨co[s], cn[s]⟩
```

We also check if the cost to m followed by P, *using the current weights* w' is shorter than any previous path (line 17). If so we update the upper bound u and the incumbent I.

Algorithm 2 simply returns the cost of the CPD path from s to t both using the original weights w and the new weights w'. The path can always be reconstructed using the CPD itself. The pseudo-code use $co[s]$ and $cn[s]$ to cache the cost of the shortest path from s to t using original weights w, and the cost of the same path, not necessarily shortest for these weights, using weights w'. We assume the cache is initialised to ∞ for each new $\mathsf{CPD\text{-}Search}$ call.

Note that because of caching of calls to wCPD, the CPD is only called for any node m for current target t at most once. Hence the CPD heuristic is amortised $\mathcal{O}(1)$ cost to compute.

Proposition 1. *Let P be the path returned by $\mathsf{CPD\text{-}Search}(s,t,w,w',\epsilon)$ then $c(P,w') \leq \epsilon c(S,w')$ for all paths S in G from s to t.*

Proof. Since the CPD heuristic is admissible, we know that $f[n]$ is no greater than the cost of the shortest path from s to t via n. When the node n in *open* with minimum $f[n]$ is such that $\epsilon f[n] \geq u$, where u is the cost of the incumbent, then we have that no path from s to t via n can cost less than u. □

6 Anytime Search

When deliberation time is limited, an optimal algorithm such as A* may not find a path in time, even with the help of sophisticated heuristics. Bounded suboptimal or bounded cost algorithms are better suited for such settings but still depend on the user to fix an acceptable quality threshold a priori. On the one hand, if the user bound is too loose the quality of the first solution may not be very good. On the other hand, if the user bound is too tight, the algorithm may not finish in time.

Anytime algorithms are a class of related methods which seek to find a first solution quickly and then continue searching, improving the incumbent until time runs out. Current state-of-the-art methods of this type include Anytime Weighted A* (AWA*) [Hansen and Zhou, 2007] and Anytime Repairing A* [Likhachev *et al.*, 2008]; both of them can be seen as variants of bounded-suboptimal search: i.e. they seek in the available time a suboptimal solution with the tightest possible suboptimality bound. A different and also state-of-the-art method is Anytime Non-parametric A*, sometimes called Anytime Potential Search [Stern *et al.*, 2014]. This method can be seen as a variant of bounded-cost search: i.e it seeks in the available time a suboptimal solution with cost less than the smallest possible cost limit C.

We can modify Algorithm 1 to be *anytime* by simply outputting a new incumbent when discovered (line 18). We can similarly extend an anytime search such as AWA* by simply using the CPD heuristic and making use of any incumbents it finds, we call this Anytime Weighted CPD-Search. The CPD heuristic offers considerable advantages for anytime search. We discuss several of them that hold for both bounded-cost and bounded-suboptimal settings.

Fast feasible plans. When available time is small, any solution at all can be acceptable. In this case anytime search can stop at the first expanded node where the CPD path is feasible. Often this node is the start node.

Strong lower-bounds. Bounded search algorithms typically proceed in best-first fashion and are guided by specialised heuristics that need to accurately estimate the cost-to-go. As a general rule, stronger lower-bounds lead to more efficient search. Because CPD heuristics are derived from concrete paths, the cost estimates account for topological features of the map such as obstacles.

Strong upper-bounds. In bounded search each new incumbent solution improves the upper-bound u and provides an opportunity to prune the search: any node n with $f(n) \geq u$ need never be expanded or generated since it cannot possibly yield a better path. CPD heuristics provide upper and lower bounds at each node, which means we can update u sooner.

Early termination. Also valuable for anytime algorithms.

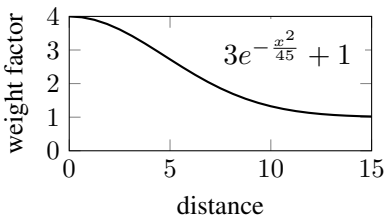

Figure 2: Weight update function for an edge at x hops (i.e., distance, in edges) from the "problem" node n.

7 Experiments

We evaluate CPD-Search in two online settings: optimal search and anytime search. In both experiments we use **grid benchmarks** drawn from Sturtevant's well known repository at http://movingai.com. Each benchmark comprises a set of grid maps and each map is associated with a set of instances (start-target pairs). They are as follows:

DAO: Drawn from the game DRAGON AGE: ORIGINS, this benchmark is representative of uniform-cost game maps.

WC3: Drawn from the game WARCRAFT III, this is benchmark is representative of weighted-cost game maps. Each map features several types of terrains to which we assign costs as follows: Ground 1.0, Trees 1.5, Shallow water 2.0, Water 2.5. When traversing from one tile to another we weight the base move cost (1 or $\sqrt{2}$ respectively) by the average terrain cost of the origin and destination tile.

MAZES: This classic benchmark consists of automatically generated labyrinths of size 512×512. Each map is uniform cost and features corridors of fixed width $k \in \{1, 2, 4, 8, 16, 32\}$. Mazes are especially challenging for search algorithms using distance-based heuristics.

ROOMS: This synthetic benchmark features uniform-cost maps with square rooms of size $k \times k$, $k \in \{8, 16, 32, 64\}$ and which are randomly connected by entrances of width 1. Room maps are designed to force the expansion of seemingly promising nodes that cannot appear on any optimal path.

In our online experiments we allow grid map traversal costs to change between different instances. We model these changes using one of two **perturbation policies**. They are:

AREA: This policy is meant to simulate a traffic problem, where some node is the location of the "problem" and edges around it are penalised with decreasing multiple less as their hop distance grows from the "problem". For each query of a path from a node s to a target node t that we perform in the experiment, a node n on the CPD path P from s to t is randomly chosen. We increase the weight of all edges in the graph within radius r of node n by an amount that decays as the distance from n increases. We use a radius $r = 15$ and a weight multiplication of $w'(e) = w(e) \times (3e^{-\frac{x^2}{45}} + 1)$ where x is the number of hops from node n to edge e. This Gaussian-like distribution is illustrated in Figure 2.

RANDOM: This policy involves a randomly selected subset of edges $E' \subseteq E$ s.t. $|E'|/|E| = 0.1$ whose weights we multiply by a factor of 3.

Benchmark	# Maps	CPD Seconds	CPD MB	Landmarks [12] Seconds	Landmarks [12] MB
DAO	156	47.29	25.86	0.03	1.79
WC3	36	721.13	364.36	0.24	9.01
Mazes	60	555.98	28.58	0.38	9.98
Rooms	40	878.64	178.26	0.42	11.28

Table 1: Preprocessing cost. We measure average time and space costs for CPD and Landmark [12] heuristics.

We undertake **empirical comparisons** between CPD-Search and ALT [Goldberg and Harrelson, 2005], a pathfinding algorithm that combines of A* search with the preprocessing-based heuristic known as *Landmarks*. We implement ALT with different numbers of landmark nodes, up to diminishing returns: 6, 12 and 18. Our test machine is a i7-8700 machine with 16GB memory. All codes are in C++ and available from https://bitbucket.org/koldar/astar-early-stop/.

7.1 Experiment 1: (Offline) Preprocessing

In Table 1 we give the time and space overheads needed to compute and store auxiliary data for CPD and Landmark heuristics (the latter with 12 nodes; results for 6 and 18 nodes are similar). Clearly CPD costs are substantially larger than landmarks but certainly not prohibitive. We make two observations: (i) we can easily store CPDs in main memory which is important for efficient online performance (we discuss performance in the next sections); (ii) since we assume the map is known a priori, the preprocessing overhead does not need to be amortised online. We compute auxiliary data once and we re-use it to speed up all subsequent queries.

7.2 Experiment 2: (Online) Optimal Grid Search

In this experiment we apply to each instance from every benchmark the `AREA` perturbation policy. Once the graph is perturbed, we solve the modified instance to optimality with both CPD-Search and ALT. The graph is then reset.

We begin with Figure 3 which makes a detailed CPU-time comparison on three selected maps from games and mazes. Each curve is a cactus-plot that represents one algorithm with times being independently sorted from smallest to largest; i.e. we show the entire distribution of results and the x-axis represents specific *instance-quantiles*. Notice from the figure that when landmarks provide accurate distance estimates, such as on the map `hrt201n`, the advantage of CPD-Search is small. The Landmark heuristic is known to be a one-dimensional estimator [Rayner *et al.*, 2011]: i.e. each landmark can be thought of as the origin of a one-dimensional Euclidean space in which all other nodes are placed according to their recorded distance from the origin. The maps `dustwallowkeys` and `maze512-1-4` are more complicated environments where one-dimensional embeddings are insufficient to provide accurate estimates between any pair of states. In these cases we see a strong advantage for CPD-Search. Table 2 gives a more comprehensive report with summary statistics for CPU time and node expansions across all benchmarks. We observe that CPD-Search strongly outperforms ALT and often by several factors.

7.3 Experiment 3: (Online) Anytime Search

In time-constrained applications (e.g., computer video games) even fast algorithms may still require too much time to compute an optimal path. This motivates us to consider CPD-Search as an anytime algorithm. We compare its performance against Anytime Weighted A* with Landmark heuristics and we develop a related variant, Anytime Weighted CPD-Search. Both weighted algorithms use the suboptimality bound $\epsilon = 2$, as in [Hansen and Zhou, 2007]. We test performance on the map `hrt201n` with `RANDOM` perturbations. As we saw in the previous section, on this map CPD heuristics have only a small benefit over Landmarks. This concentrates the comparison on the anytime advantages of CPD-Search, rather than on the advantages that come from the CPD heuristic being more accurate.

In Figure 4 (left) we report normalised solution costs at various time cutoffs. We observe strong improvements for CPD-Search. For example, at 0.03ms AWA* with Landmarks has almost no incumbent solutions while both CPD methods have incumbents for at least 25% of instances. By 0.3ms AWA* with Landmarks has incumbents for $<$50% of instances while CPD methods have incumbents for $>$75%. More interesting is the comparison between anytime CPD-Search and its use in AWA*. Clearly Anytime Weighted CPD-Search is slightly better at generating good solutions early. However by 1ms CPD-Search has a better median value and is only slightly worse for max and mean. By 4ms CPD-Search has proven optimality for all instances and is finished. Neither AWA* variant reaches this point, even by 10ms. These results clearly shows that CPD-Search is a competitive anytime algorithm, even without modification.

In Figure 4 (right) we report performance for optimal CPD-Search and optimal A* with Landmarks [12]. Notice that median values are only 1ms and 1.5ms respectively; i.e., the only anytime behaviour that is interesting is before that. The plot also reinforces the benefit of CPD-Search over Landmarks, this time with `RANDOM` cost perturbations.

8 Conclusions

Compressed Path Databases (CPDs) are a family of ultra-fast pathfinding algorithms which eliminate the need for runtime search in static-cost maps. In this work we propose CPDs as heuristic functions for dynamic settings where costs can only increase. We describe a new algorithm, CPD-Search, which is the combination CPD heuristics with A*. In a first experiment we use standard grid benchmarks to show that CPD-Search strongly outperforms ALT [Goldberg and Harrelson, 2005], a well known and state-of-the-art method. In a second experiment we consider CPD-Search as an anytime algorithm. Here we find that it generates good solutions earlier and optimal solutions faster than even dedicated methods such as Anytime Weighted A* [Hansen and Zhou, 2007].

Our results are based on SRC [Strasser *et al.*, 2015], a leading CPD solver from the 2014 Grid-based Path Planning Competition. Since 2014 several works extend SRC by improving its compression strength [Salvetti *et al.*, 2017], its running time [Salvetti *et al.*, 2018] or both [Chiari *et al.*, 2019]. These techniques can be easily combined with CPD-Search to further improve current results.

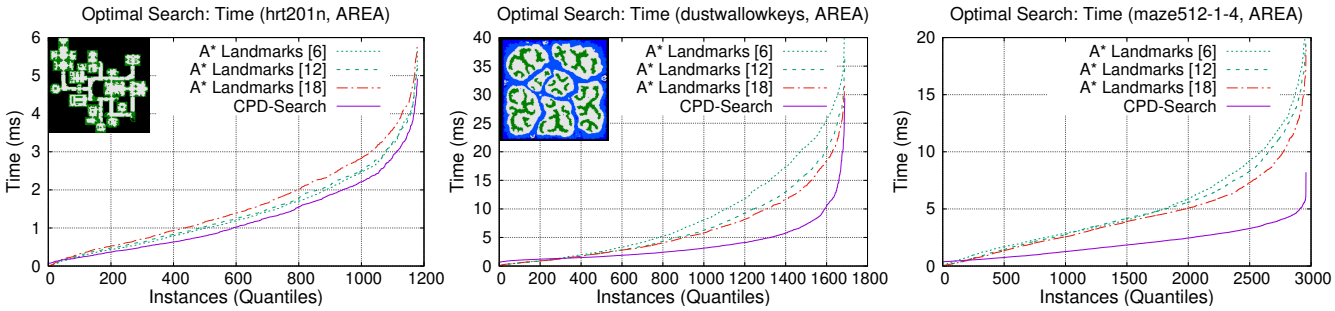

Figure 3: CPU time performance (y-axis, in ms) for selected grid maps. Each curve sorts the set of instances (x-axis) monotonically. We compare CPD-Search against A* with the Landmark heuristic (in square brackets is the number of landmarks).

Benchmark	# Instances	Algorithm	Time (ms)				Expanded Nodes			
			Q1	median	Q3	mean	Q1	median	Q3	mean
DAO	154,340	A* Landmarks [6]	1.28	4.24	13.14	9.55	2687	9447	30419	22549
		A* Landmarks [12]	1.26	4.47	15.12	10.84	2095	7958	28922	21177
		A* Landmarks [18]	1.31	4.67	16.63	12.09	1848	7197	27704	20528
		CPD-Search	**0.68**	**1.54**	**3.71**	**3.23**	**515**	**1668**	**4759**	**4048**
WC3	55,460	A* Landmarks [6]	2.84	9.03	25.38	17.32	5310	18751	54492	37109
		A* Landmarks [12]	2.02	5.77	15.39	13.39	2904	9544	26791	23044
		A* Landmarks [18]	1.69	4.43	10.32	9.99	1924	6031	15166	14705
		CPD-Search	**1.22**	**2.03**	**3.92**	**3.11**	**335**	**1246**	**3533**	**2603**
Mazes	61,960	A* Landmarks [6]	3.10	6.97	13.80	9.92	7047	16819	33421	23669
		A* Landmarks [12]	3.21	7.07	14.04	10.34	5775	13239	26449	19276
		A* Landmarks [18]	3.45	7.44	14.91	11.00	5253	11811	23829	17540
		CPD-Search	**1.56**	**3.27**	**7.97**	**6.47**	**1560**	**5048**	**13085**	**9965**
Rooms	41,060	A* Landmarks [6]	3.01	8.73	21.59	15.93	5760	18810	48275	34270
		A* Landmarks [12]	2.42	6.16	13.65	10.78	3610	10779	25390	20139
		A* Landmarks [18]	2.25	5.36	11.16	7.87	2799	7955	17622	12197
		CPD-Search	**1.30**	**2.14**	**4.12**	**3.62**	**553**	**1660**	**4379**	**3716**

Table 2: Optimal Search with AREA perturbations. We compare CPD Search with A* Landmarks on 4 grid benchmarks. We report results for (CPU) Time and Expanded Nodes. Q1 and Q3 indicate the 1st and 3rd quartiles of each distribution.

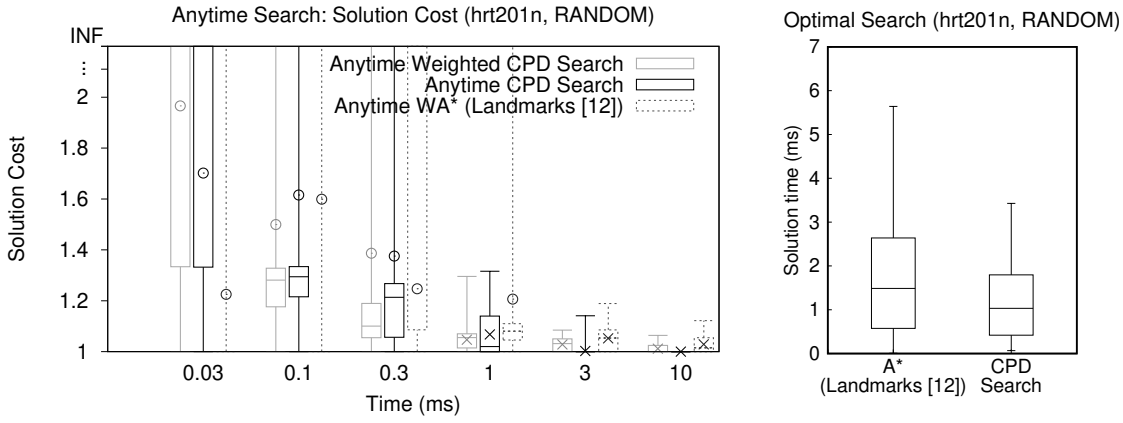

Figure 4: Anytime and Optimal Search with RANDOM perturbations. (Left) We show the distribution of best solution costs (normalised by optimal cost) at different time cutoffs. A cross on each box-plot indicates average (normalised) solution cost (where defined). When the max is infinite a circle represents the largest non-infinity normalised solution cost. (Right) We report (CPU) time distributions for optimal search on hrt201n.

References

[Abraham *et al.*, 2012] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Hierarchical Hub Labelings for Shortest Paths. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Proceedings*, pages 24–35, 2012.

[Botea and Harabor, 2013] Adi Botea and Daniel Harabor. Path Planning with Compressed All-pairs Shortest Paths Data. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 293–297, 2013.

[Botea, 2011] Adi Botea. Ultra-Fast Optimal Pathfinding without Runtime Search. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011*, pages 122–127, 2011.

[Chiari *et al.*, 2019] Mattia Chiari, Shizhe Zhao, Adi Botea, Alfonso Gerevini, Daniel harabor, Alessandro Saetti, Matteo Salvetti, and Peter J. Stuckey. Cutting the Size of Compressed Path Databases With Wildcards and Redundant Symbols. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 2019.

[Delling and Wagner, 2007] Daniel Delling and Dorothea Wagner. Landmark-Based Routing in Dynamic Graphs. In *Experimental Algorithms, 6th International Workshop, WEA 2007, Proceedings*, pages 52–65, 2007.

[Delling *et al.*, 2011] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato Fonseca F. Werneck. Customizable Route Planning. In *Experimental Algorithms - 10th International Symposium, SEA 2011, Proceedings*, pages 376–387, 2011.

[Delling *et al.*, 2017] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning in Road Networks. *Transportation Science*, 51(2):566–591, 2017.

[Dibbelt *et al.*, 2014] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. In *Experimental Algorithms - 13th International Symposium, SEA 2014, Proceedings*, pages 271–282, 2014.

[Funke and Storandt, 2015] Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 45:1–45:10, 2015.

[Geisberger *et al.*, 2008] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Experimental Algorithms, 7th International Workshop, WEA 2008, Proceedings*, pages 319–333, 2008.

[Geisberger *et al.*, 2012] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

[Goldberg and Harrelson, 2005] Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2005.

[Hansen and Zhou, 2007] Eric A. Hansen and Rong Zhou. Anytime Heuristic Search. *Journal of Artificial Intelligence Research*, 28:267–297, 2007.

[Koenig *et al.*, 2004] Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy. Incremental Heuristic Search in AI. *AI Magazine*, 25(2):99–112, 2004.

[Kring *et al.*, 2010] Alexander William Kring, Alex J. Champandard, and Nick Samarin. DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010*, pages 39–44, 2010.

[Likhachev *et al.*, 2008] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artif. Intell.*, 172(14):1613–1643, 2008.

[Rayner *et al.*, 2011] D. Chris Rayner, Michael H. Bowling, and Nathan R. Sturtevant. Euclidean Heuristic Optimization. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, pages 81–86, 2011.

[Salvetti *et al.*, 2017] Matteo Salvetti, Adi Botea, Alessandro Saetti, and Alfonso Gerevini. Compressed Path Databases with Ordered Wildcard Substitutions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 250–258, 2017.

[Salvetti *et al.*, 2018] Matteo Salvetti, Adi Botea, Alfonso Emilio Gerevini, Daniel Harabor, and Alessandro Saetti. Two-Oracle Optimal Path Planning on Grid Maps. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 227–231, 2018.

[Schultes and Sanders, 2007] Dominik Schultes and Peter Sanders. Dynamic Highway-Node Routing. In *Experimental Algorithms, 6th International Workshop, WEA 2007, Proceedings*, pages 66–79, 2007.

[Stern *et al.*, 2014] Roni Stern, Ariel Felner, Jur van den Berg, Rami Puzis, Rajat Shah, and Ken Goldberg. Potential-based bounded-cost search and Anytime Non-Parametric A*. *Artif. Intell.*, 214:1–25, 2014.

[Strasser *et al.*, 2015] Ben Strasser, Adi Botea, and Daniel Harabor. Compressing Optimal Paths with Run Length Encoding. *Journal of Artificial Intelligence Research*, 54:593–629, 2015.

[Sturtevant *et al.*, 2009] Nathan R. Sturtevant, Ariel Felner, Max Barrer, Jonathan Schaeffer, and Neil Burch. Memory-Based Heuristics for Explicit State Spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 609–614, 2009.