Optimal Graph Search with Iterated Graph Cuts

David Burkett **David Hall** **Dan Klein**
Computer Science Division
University of California, Berkeley
{dburkett,dlwh,klein}@cs.berkeley.edu

Abstract

Informed search algorithms such as A* use heuristics to focus exploration on states with low total path cost. To the extent that heuristics underestimate forward costs, a wider cost radius of suboptimal states will be explored. For many weighted graphs, however, a small distance in terms of cost may encompass a large fraction of the unweighted graph. We present a new informed search algorithm, Iterative Monotonically Bounded A* (IMBA*), which first proves that no optimal paths exist in a bounded cut of the graph before considering larger cuts. We prove that IMBA* has the same optimality and completeness guarantees as A* and, in a non-uniform pathfinding application, we empirically demonstrate substantial speed improvements over classic A*.

Informed search algorithms such as A* (Hart, Nilsson, and Raphael 1972) have seen success in a wide range of applications, including planning (Hoffmann and Nebel 2001), robot navigation (Koenig and Likhachev 2002), vehicle routing (Gendreau, Hertz, and Laporte 1994), and even natural language parsing (Klein and Manning 2003). A* and its cousins all employ a heuristic, which is a function that gives an approximation (typically a lower bound) of the remaining distance from each state to the goal state. These algorithms combine information about both exact costs from the start state and the approximate costs to the goal state to avoid exploring states that are not on the optimal path to the solution.

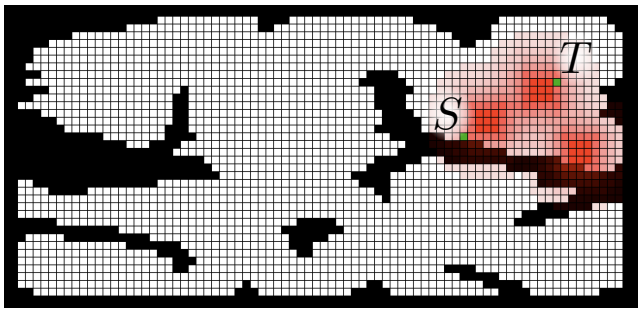
However, the performance of informed search algorithms depends on the quality of the heuristic function: if the heuristic provides a tight estimate of the remaining distance to the goal, then these algorithms can avoid exploring much of the state space. On the other hand, if the bound is loose, then these algorithms will often cover a large portion of the low cost region of the state space. Indeed, if the bound is too loose, they end up faring no better than uninformed algorithms like uniform cost search.

In this paper, we are specifically concerned with search on graphs that contain high cost regions whose structure cannot be easily captured by a heuristic. For example, in pathfinding applications, there might be areas that are more difficult to traverse but that cannot be characterized without solving the search problem. To the extent that these high cost areas surround the goal and start states, algorithms that rely only on projections of forward costs to guide the search, such as A*, will explore large low cost portions of the search space before finally traversing the unavoidable high cost regions.

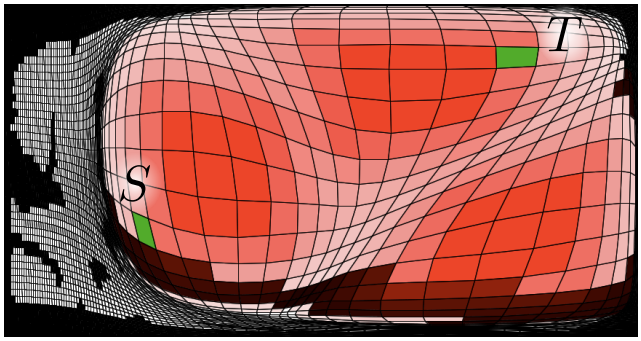
Of course, one way to fix this problem is to somehow improve the quality of the forward cost heuristics. This work takes an orthogonal approach. We describe an algorithm that performs a sequence of searches on restricted cuts of the graph, stopping as soon as it finds a path that is provably optimal in the original graph. The algorithm is complete and optimal, and in our implementation, called Iterative Monotonically Bounded A* (IMBA*), we use A* as the inner search procedure, which lets IMBA* also incorporate the benefits of heuristic guided search.

There are other algorithms that manipulate the search graph to improve speed. For example, Hierarchical A* uses a sequence of carefully-defined graph projections to define a cascade of search problems, where the solution for one problem is used as the heuristic for the next (Holte et al. 1996). Another approach involves the definition and composition of abstraction operators that typically correspond to subproblems of the original task (Helmert, Haslum, and Hoffmann 2007; Yang et al. 2008; Katz and Domshlak 2010). These approches differ from ours in two major ways. First, they both require a natural decomposition of the graph into subtasks or subregions, which may not always be feasible. Second, they are both methods for computing improved heuristics, with the final search still performed on the original graph. However, in search problems where they apply, they could be used as part of the inner search for IMBA*, possibly stacking the benefits of each approach.

Looking only at other algorithms that use a modified search rather than an improved heuristic, the algorithms most closely related to IMBA* are the bidirectional variants of A* (Pohl 1969). Like IMBA*, these algorithms explore the expensive points around both start and goal quickly, avoiding the problems characteristic of A* in these environments. However, straightforward implementations do not preserve the optimality guarantees of A*. Instead, one must run the search algorithm until a specific condition is reached (Kaindl and Kainz 1997); merely connecting the two fringes is not sufficient for optimality. In our experiments, we show that IMBA* is faster than even the non-



(a) Unweighted



(b) Weighted

Figure 1: A standard grid search problem drawn to scale according to unweighted graph distance (a) and weighted distance (b). High-traversal-cost regions are shown in red.

optimal variant of bidirectional A* in threat-aware pathfinding scenarios commonly seen in real-time strategy games.

IMBA*

Motivation

First, we present a caricature of the kind of search problem we are interested in. Consider the weighted grid problem in Figure 1a. Here we are seeking a minimum cost path from the start state S to the goal state T. In the unweighted graph, S and T are quite close, but in weighted space (Figure 1b), they appear to be far away. Indeed, a uniform cost search of this graph would exhaustively explore the region "behind" S before finally traversing the close, but high cost, path to T.

We would like to use an informed search algorithm like A* to improve upon the performance of this uniform cost traversal. Unfortunately, without advance problem-specific knowledge of the structure of the graph (e.g. where high cost regions are), the only heuristic we can easily specify is unweighted graph distance. However, this heuristic grossly underestimates the distance to the goal, and so in this case A* would not perform much better than uniform cost search.

What went wrong here? A* prefers to expand states that make progress toward the goal, but because A* must return optimal solutions, it must be cautious: it might be the case that there is some minimum cost path that goes exclusively through the low cost region, and so A* tolerates some deviation from making progress towards the goal (in terms of heuristic cost) in order to possibly find a lower cost path.

However, as implementors we know—or at least suspect—that these deviations will not actually find the optimal path, and that any path to the goal will likely not stray too far from the start and goal states. A naive solution would be to simply restrict the search space to those states that are somehow "near" either the start or the goal. We could define a cut of the graph that includes both the start and the goal, and attempt to find a path that remains inside that cut. Of course, if we simply delete states and edges, we might remove the optimal path or even all paths. In the latter case we could simply take larger and larger cuts until we find a path, but the former issue is more difficult to address: when we find a path in a cut how do we know that it is optimal in the full graph?

Iterated Bounded Cuts

To answer this question, we introduce Iterative Monotonically Bounded A* (IMBA*). The core improvement to the naive graph-cut approach above is that each cut is repaired in such a way that the connectivity of the original graph is preserved: for all states in the cut with a path between them in the original graph, there is some path between them in the repaired cut graph with cost less than or equal to the path in the original graph. We formalize this notion later, but first we present an execution of the algorithm in a small grid.

Consider the search problem in Figure 2a. As in Figure 1, the optimal path from the start to the goal must pass through a high cost region. Moreover, the optimal path never strays far from a small area around the start and the goal. Therefore, we place a bet that the optimal path will not leave the area marked by a blue rectangle in Figure 1b. To ensure that a lower bound on the optimal path is preserved, we add or adjust edges between the states on the border (highlighted in purple) between the cut and the exterior of the graph. Here, we make all of these new edge costs 1, since that is the minimum cost between any two states on the original graph. In general, we only add edges that create paths of *lower* cost than existing paths through the exterior of the graph.

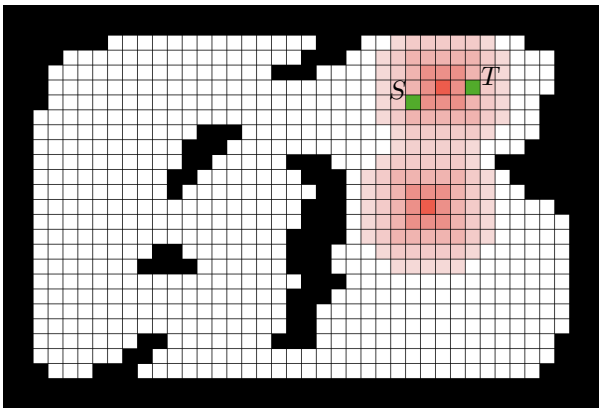
Given this repaired cut, we simply find the shortest path from S to T in this graph, shown in Figure 1b. Because this path goes through the border, it could be the case that the optimal path in the full graph lies outside this cut, as indeed it does. Therefore, we *relax* the cut, taking a larger portion of the graph and repairing the cut as before (Figure 1c). Now the shortest path in this graph lies entirely on the *interior* of the cut. Therefore, we know that the path is optimal in the whole graph, because any path that goes through the exterior of the graph has some path that is cheaper that passes through the border region.

Thus, our algorithm has managed to limit its exploration to a relatively small fraction of the search space, whereas standard A* in this case explores 94% of the grid.

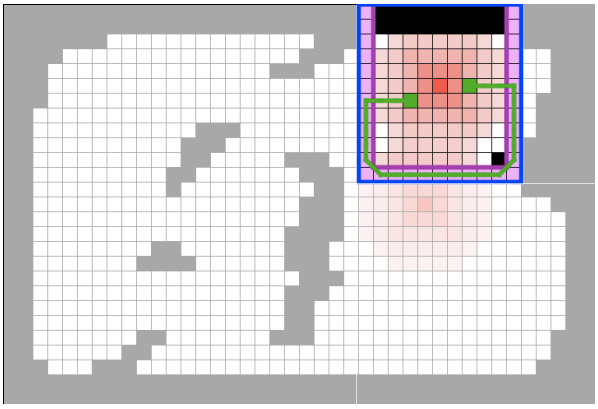
Algorithm

Formally, define a search problem on a weighted graph G with a set of states V and edges E. We seek a path from the start state S to goal state T.

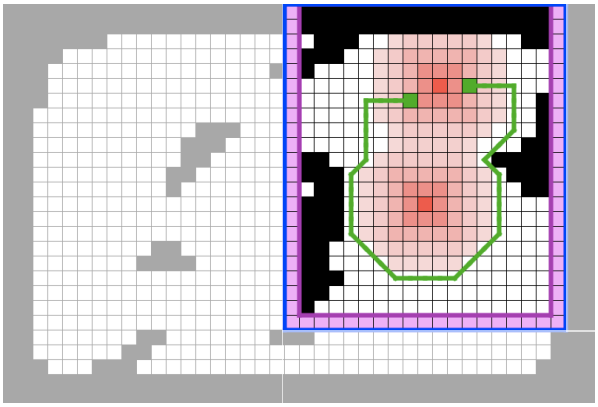
Now define a sequence of N cuts $\{S, T\} \subseteq C_1 \subset C_2 \subset$



(a) Grid Search Problem



(b) First Cut



(c) Second Cut

Figure 2: A full grid search problem (a) and two successive searches, restricted to a smaller (b) and slightly larger (c) region of the grid. The boundary of each restricted set, where all obstacles are removed and traversal costs are reduced to 1, is shaded purple. Start and goal states, and the optimal paths between them, are shown in green.

Algorithm 1 IMBA*

Require: A sequence of cuts C_i
Require: A start state S and a goal state T
Require: An optimal search procedure and a repair procedure
  for $i = 1 \cdots$ **do**
    $C_i^+ \leftarrow \text{repair}(C_i)$
    path,cost $\leftarrow \text{search}(C_i^+, S, T)$
    if cost $= \infty$ **then** {no path exists}
      return path,cost
    else if $\neg\forall n \in \text{path}.n \notin B_i$ **then** {optimal path found}
      return path,cost
    end if
  end for

$\cdots \subset C^N = V$.[1] The *exterior* X_i of a cut is $G \backslash C_i$. The *border* $B_i \subseteq C_i$ of a cut is the set of states that share at least one edge with a state in X_i (the purple edges in Figure 3a). In a grid, this border really will appear to be a visible "border" of the cut, though in a more general graph it might not be.

Given a cut C_i and its border B_i we define a *repaired* cut C_i^+ as any graph with the following properties:

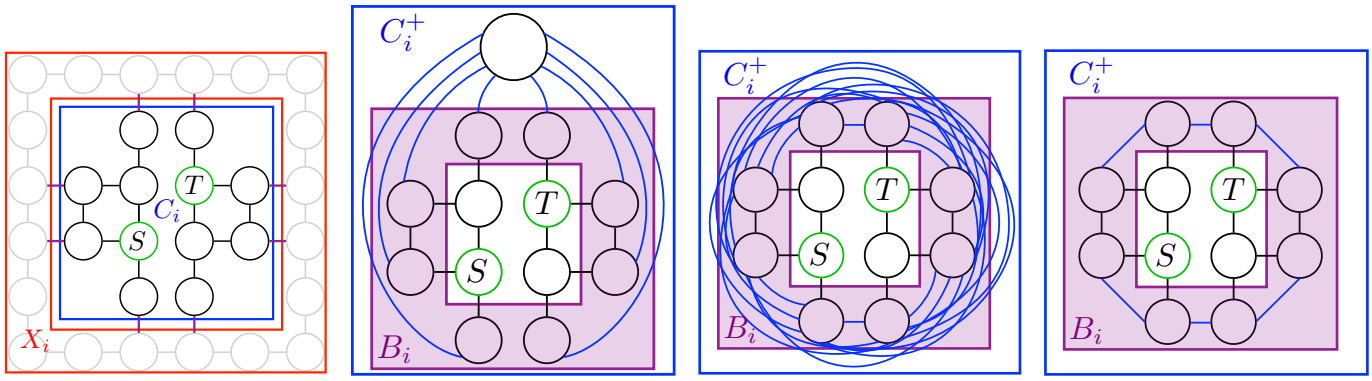
1. *Node Preservation:* The set of states in C_i^+ is a superset of the states in C_i.
2. *Edge Preservation:*Any edge directly connecting states in C_i is preserved.
3. *Interior Maintenance:* No edges may be added that connect states that lie on the interior $C_i \backslash B_i$ of the graph.
4. *Path Optimism:* If there exists a path in G connecting two states in B_i that passes through X_i with cost c, then there must exist a path in C_i^+ between those same states with cost $c' \leq c$.

These properties ensure that all paths in the original graph have some analog in the cut graph of the same or cheaper cost. We will use these properties to prove the correctness of our algorithm.

With the notation above, the algorithm is quite easily specified (Algorithm 1). Essentially, we iterate through each cut C_i of the graph, repair that cut to form C_i^+, and run an optimal and complete search algorithm from the start state to the goal state in C_i^+. If the search procedure does not find any path in a cut, then the entire algorithm can terminate, since there must be no path at all in the original graph. That is, IMBA* can "fail fast" without exploring the entire state space, something that typical A* cannot do. (See Figure 4 for an example.) If instead a path is found that does not pass through the border B_i, then that path is also optimal in G, and it can be returned. Otherwise, we proceed to the next cut. Since the last cut is equal to the entire graph, the algorithm will terminate.

It is worth stressing that—although we term our algorithm IMBA*—it is actually not required that the inner search pro-

[1] It is actually not necessary but merely sufficient for the cuts to be subsets of each other. For clarity, we do not attempt a definition of the exact conditions.



(a) Graph Cut (b) Extra Node Repair (c) All Edges Repair (d) Natural Edges Repair

Figure 3: A cut of a search graph (a) that leaves no path from S to T, and three different methods of repairing the cut (b-d). Edges introduced as part of the repair are shown in blue.

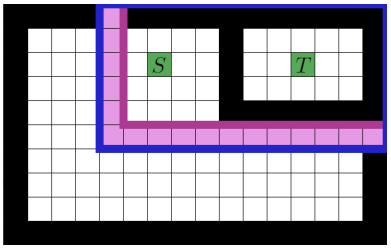


Figure 4: A search problem with no feasible solution. Standard algorithms such as A* would expand the entire graph in this instance, while IMBA* will terminate after trying the outlined cut.

cedure be A*. Indeed, it can be any optimal and complete search algorithm. Thus, any of the many variants of A*—e.g. Iterative Deepening A* (Korf 1985)—or even uniform cost search can be used.

Correctness

In this section we prove that IMBA* is both complete and optimal; that is, that it always returns a path if there is one, and that it always returns the best possible path. The proofs are straightforward.

Proposition 1. *IMBA* is complete.*

Proof. At some point the succession of cuts $C_i \cdots C_N$ will contain any path (by construction) in the underlying graph, since the cuts must grow towards including all states. Thus, the only concern is that IMBA* aborts early, which it can do for one of two reasons: either it returns no path, or it returns a path that does not correspond to some path in the full graph G.

In the first case, it could be that in some cut C_i^+, there is no path from S to T, but there is such a path in the entire graph G. Since C_i^+ contains all paths that lie entirely within C_i (Node and Edge Preservation), this path must both exit and enter the exterior X_i of the graph, and therefore must pass through at least two boundary states. However, by construction, C_i^+ has a path between those two states (Path Optimism), meaning that there must be some path in C_i^+. Thus, we have a contradiction.

In the second case, IMBA* could return a path that does not exist in the true graph. To do so, it would have to use an edge that does not exist in the underlying graph G. Since no edges can be added entirely within the interior of the cut (Interior Maintenance), the path must touch a border state, contradicting our assumption that IMBA* terminated early. □

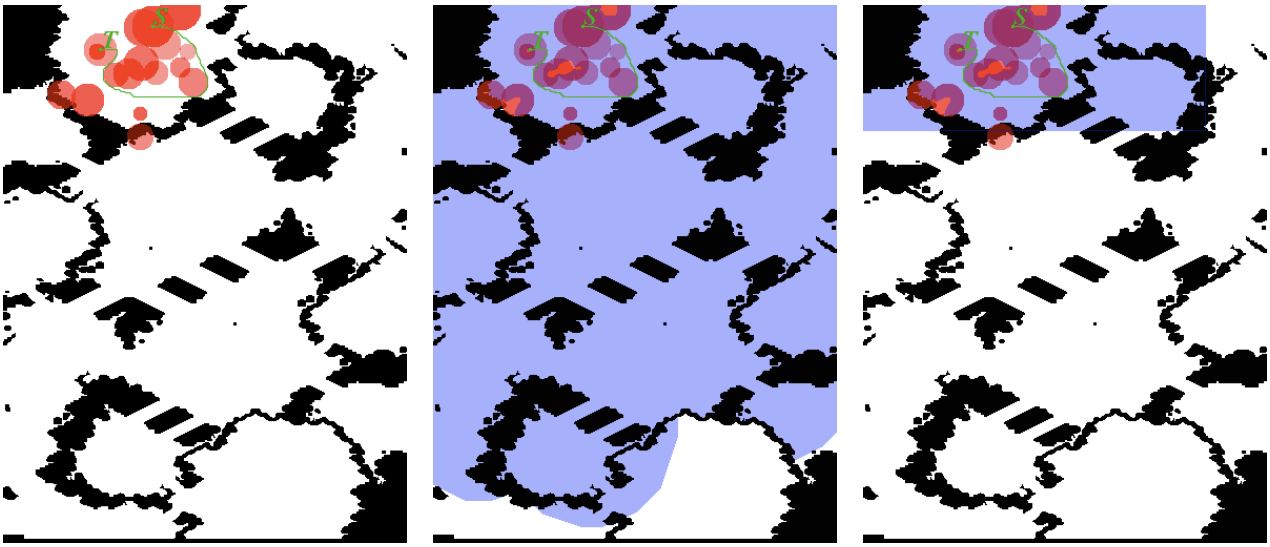
Proposition 2. *IMBA* is optimal.*

Proof. Suppose that the algorithm returned a suboptimal path when searching in some cut C_i^+. By our requirement on the underlying search procedure, any path found in C_i^+ must be optimal for C_i^+. Moreover, from above we know that this path must correspond to some path in the true graph and therefore consist only of edges in the true graph. (Otherwise, it would have to pass through a border state, and IMBA* would not return it.) This path has cost c which is less than all other path costs in C_i^+. We also know that the true optimal path (with cost c^*) does not lie entirely within the cut, because otherwise the underlying search would have found it. Thus, this path has some corresponding path in the repaired cut graph that passes through the border region with cost $c' \leq c^*$ (Path Optimism). But this means that $c' \leq c^* < c$, contradicting our assumption that c is the cost of the optimal path in C_i^+. □

Defining and Repairing Cuts

We have largely left open two key pieces of the algorithm: defining and repairing cuts. In this section we discuss a few approaches and considerations when specifying these. Largely, these points are guidelines, and it will fall to the implementor to find and test cuts that are well-suited to the particular domain.

Defining the cuts themselves typically requires a certain amount of domain knowledge. In the case of pathing problems, a simple bounding box construction around the start



(a) Example Search Problem (b) Nodes Expanded by A* (c) Nodes Expanded by IMBA*

Figure 5: An example search problem from our evaluation set, including the optimal path from S to T. In (b) and (c), the nodes highlighted in blue are those expanded by A* and by IMBA*, respectively.

and goal state seems to work well, as we've shown in the examples. In more general graphs, cutting based on a measure of graph distance from the start and goal state might be appropriate. One simple extension is to take a cut that includes an unweighted shortest path in the underlying graph. Areas around this path may contain the optimal path, though of course this will depend on the particular search problem.

Defining a good sequence of cuts is perhaps the trickiest aspect of the algorithm. If the cuts are too tight or grow too slowly, then IMBA* will waste a significant amount of time re-searching over the same area. If, on the other hand, they grow too quickly, then the algorithm may perform no better than standard A*. In our particular application, we found that growing the bounding boxes by 4x (doubling both their width and their height) was effective.

Finally, one has to choose a method for repairing a cut C_i into C_i^+. We describe three simple methods, though more complicated methods may fare better, depending on the application. Consider the graph in Figure 3a. The simplest method is to treat the exterior of the graph X_i as a single state, and add edges to each boundary state into and out of X_i as appropriate. (See Figure 3b.) Alternatively, one can add edges directly between all boundary states, as in Figure 3c. Depending on the graph, both of these methods may exhibit branching factors that are much too high, as there is at least one state that touches every state on the border. A third approach—especially suitable for grids—is the one illustrated in Figure 3d. Here, we add arcs that are logical neighbors even if they are not connected in the original graph. For all of these methods, the weights of the edges can be set to 0, though any weights that preserve the lower bound property work and higher weights will perform better.[2]

[2]While IMBA* does not require any further properties, the choice of edge weights may interact with the heuristic used for the inner A* procedure. Namely, one must take care to preserve consistency.

Algorithm	Time (ms)	Nodes Expanded	Num Cuts	Path Cost
A*	414	110,064	1.0	1.000
Bi A* (Opt)	923	214,078	1.0	1.000
Bi A* (Fast)	144	36,276	1.0	1.005
IMBA*	83	23,806	2.2	1.000

Table 1: Empirical speed results for A* vs IMBA*. All values are averaged per search problem. Path costs are scaled such that optimal paths have cost 1.0. Bi A* refers to an implementation of the algorithm of Kaindl and Kainz 1997, which has optimal and fast variants.

Experiments

The development of our algorithm was motivated by our need for an efficient pathfinder for an agent we developed for the AIIDE 2010 StarCraft AI Competition. Our agent needed to find safe paths to navigate its units into and out of enemy territory while taking a minimum of damage. The representation we chose modeled enemy units as soft obstacles, that is, as regions with high cost. Typically, our agent needed to find fairly short paths in terms of unweighted graph distance. However, these paths almost always ventured into areas heavily threatened by enemy units.

To simulate these situations, we generated 100 randomly created scenarios based on a standard StarCraft map.[3] We show an example in Figure 5a. Though StarCraft allows nearly arbitrary movement directions, we restricted these scenarios to an 8-connect grid at the finest level of terrain, giving 196,608 positions, 23% of which were impassable.

We compared IMBA* to three baselines. First was our implementation of A*. (IMBA* calls this implementation as

[3]Available at http://overmind.cs.berkeley.edu/maps.

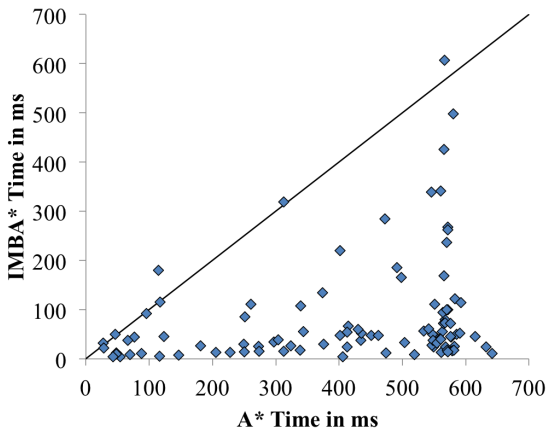


Figure 6: The runtime in milliseconds for each search problem, using either A* or IMBA*. Points along the diagonal represent problems that took the same length of time for both algorithms. In most cases, IMBA* is quite a bit faster, but there are a few problems where the search expanded to the entire grid, and the additional overhead of the earlier cuts resulted in a slight slowdown for IMBA*.

its inner search procedure.) We also tried two variants of bidirectional A* (Kaindl and Kainz 1997): one that guarantees an optimal path (Opt), and a suboptimal version that returns a path as soon as the two fringes intersect (Fast). We used Euclidean distance as our heuristic. We ran our experiments on a Large Amazon EC2 instance, and averaged over 100 runs. The results are shown in Table 1. We found that our implementation of IMBA* was on average 4-5 times faster than A*, measured both in absolute runtime and total node expansions.[4] For this class of problems, bidirectional A* does extremely poorly when forced to return an optimal path, essentially solving the search problems in both directions. When permitted to return early, bidirectional A* is much faster than A* and exhibits a relatively low degree of suboptimality, but it is still not as fast as IMBA*, which is guaranteed to be optimal.

In Figure 5b we depict graphically the state space that A* explores as compared to IMBA* in Figure 5c. A* expands all nodes that have f-cost (that is, accumulated backward cost plus estimated forward cost) less than or equal to the f-cost of the goal. IMBA*, on the other hand, explores many fewer nodes, exploring only those nodes with the same or lesser f-cost on the narrowest cut provably containing the optimal path. In fact, in this example neither A* nor IMBA* significantly benefit from the heuristic at all, exploring almost all possible low-cost nodes. IMBA*, however, does benefit from using the graph cuts, since the source and goal nodes are so close to each other.

Of course, our algorithm is not guaranteed to always outperform A*, and so we plotted the time taken by A* versus IMBA* across these executions in Figure 6. While IMBA* is usually much faster, there are a handful of search problems where A* has a slight advantage. The main shortcoming of these examples is the repeated search that IMBA* performs. We suspect that it should be possible to reuse some information from earlier cuts to speed computation of later cuts, though we leave that for future research.

[4] A node was counted twice if IMBA* expanded it as part of two successive cuts.

Conclusion

In this paper, we have presented a novel technique for efficiently finding shortest paths in graphs with high cost regions that can not be easily captured by traditional heuristic functions. Our algorithm exploits the intuition that paths should generally stay close to the start and goal states, in a way that is sometimes hard to incorporate into the design of admissible and consistent heuristics. We proved that our algorithm–using graph cuts with the right properties–preserves the optimality and completeness guarantees of A*. In simulations reflecting actual pathfinding scenarios, IMBA* significantly outperformed classical A*.

References

Felner, A.; Korf, R. E.; and Hanan, S. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134:9–22.

Gendreau, M.; Hertz, A.; and Laporte, G. 1994. A tabu search heuristic for the vehicle routing problem. *Management Science* 40(10):pp. 1276–1290.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1972. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bull.* 28–29.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *In Proc. ICAPS 2007*, 176–183.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: fast plan generation through heuristic search. *JAIR* 14:253–302.

Holte, R.; Perez, M.; Zimmer, R. M.; and Macdonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI*, 530–535.

Kaindl, H., and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *CoRR*.

Katz, M., and Domshlak, C. 2010. Implicit abstraction heuristics. *JAIR* 39:51–126.

Klein, D., and Manning, C. 2003. Factored A* search for models over sequences and trees. In *IJCAI*, 1246–1251.

Koenig, S., and Likhachev, M. 2002. Improved fast replanning for robot navigation in unknown terrain. In *ICRA '02*, volume 1, 968 – 975.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Pohl, I. S. 1969. *Bi-directional and heuristic search in path problems*. Ph.D. Dissertation, Stanford University, Stanford, CA, USA.

Yang, F.; Culberson, J. C.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *JAIR* 32:631–662.