Explicit Abstraction & Refinement

• Can’t* use implicit abstractions in graph-based domains such as pathfinding

• Build explicit abstraction
  • Pre-processing step like PDB

* and get a useful heuristic

Clique Abstraction (Sturtevant & Buro, 05)

• Abstract together nodes that form cliques
  • Maximum number of nodes within radius 1

• Works on any graph, but is optimized for 2D maps
Algorithm

- Choose group of nodes to abstract together
  - Nodes must be connected
  - Nodes cannot already be abstracted
  - Repeat until all nodes abstracted
- Add an edge between abstract groups if there exists an edge between any two nodes abstracted by each group

Properties

- An homomorphic abstraction is refineable if:
  - There is an abstract path between two nodes, there is a low-level path between all low-level nodes which abstract into the high-level nodes
  - If two nodes are connected, they share a common abstract node at some level
Approaches to Refinement-Style Search

- Find abstract path
  - Which level to start?
- Refine path
  - Complete refinement
  - Partial refinement (k-steps)
- Refinement corridor

PRA*

- PRA*
  - Complete refinement
  - Corridor fixed at 1-beyond abstract path
Pathfinding

• Given hierarchy of parents choose appropriate level to start pathfinding
• Use A* to calculate path from start to goal
Pathfinding

- Given abstract path:
  - Path defines a corridor in the lower level of abstraction
  - Run A* in this corridor to find next path
  - Repeat until done
Mathematical Analysis

- Why use the middle level?
- Suppose $k$ nodes are abstracted together at each level, $N$ nodes at level 0
- If there are $d$-levels of abstraction, $k^d = N$
- For what value of $x$ is $k^x = \sqrt{N}$?
  - $k^x = N^{0.5} = x \log k = 0.5 \log N = 0.5 d \log k$
  - $x = 0.5 d$
Suboptimality

- Worst-case suboptimality?
  - Assume abstract edges are cost 1
  - $2^d$ nodes abstracted into a node at height $d$
  - Path length through the node might be 1 but might also be $2^d$
  - A path length $L$ in abstract space could be $L$ or $2^d$ in the real space
  - Potential for exponential suboptimality

**Predicting Work (Holte, 96a)**

- Definitions:
  - $n$ - the number of nodes abstracted together
  - $d$ - the maximum diameter of an abstract node
- Holte’s analysis suggested:
  - Maximize $n$ - decrease # of refinement steps
  - Minimize $d$ - decrease cost of refinement
Generalization

- Ad-hoc ordering for nodes:
  - **Edge difference**: how many edges are removed/introduced when $n$ is contracted
  - **Original edges**: how many edges have already been abstracted below shortcut edges introduced when $n$ is contracted
  - **Upward path length**: max of unpacked path length
  - **Contracted neighbors**: how many neighbors are already contracted

Building Contraction Hierarchies

- Choose most important node $n$
  - For all pairs of neighbors, check if removing $n$ influences the shortest path between neighbors
    - If not, just remove $n$
    - If so, add shortcut edge with the same cost as the shortest path through $n$

Node Importance

Using CHs
Each node, because the computation of an optimal ordering reduce the search space. Consider the query between the leftmost and rightmost node. The CH query will visit nodes 4 is more important, so only that node will be expanded. Node 4's only unexplored neighbor is node 5, so the shortest contracting every other node requires CH and allows different tradeoffs between time and space. Figure 6: Two possible CHs from the input graph. The nodes are labeled with their importance and accordingly vertically aligned.

Reducing Contraction Hierarchy Overhead

<table>
<thead>
<tr>
<th>Using CHs</th>
<th>Using CHs</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="#" alt="Graph 1" /></td>
<td><img src="#" alt="Graph 2" /></td>
</tr>
<tr>
<td><img src="#" alt="Graph 3" /></td>
<td><img src="#" alt="Graph 4" /></td>
</tr>
<tr>
<td><img src="#" alt="Graph 5" /></td>
<td><img src="#" alt="Graph 6" /></td>
</tr>
<tr>
<td><img src="#" alt="Graph 7" /></td>
<td><img src="#" alt="Graph 8" /></td>
</tr>
</tbody>
</table>

We use a priority queue whose minimum element contains the best node to contract next, and then successively contract attractive it is to contract it next. Initially, this term is computed for each node of the graph are visited. Figure 6(c), on the other hand, requires 2 shortcuts. But the same query will generate the actual path takes for each uncontracted neighbor of the contracted node, but not for the important endpoint, as the search only progresses upwards. This term works to keep the original edges contracted into new shortcut edges. The third term is introduced when contracting an an edge. If a new shortcut is the number of shortcuts added, the last term measures the number of original edges contracted into new shortcut edges. We define for each uncontracted neighbor, we update the best node to contract next. Formally, we use a block representation for DAO. Instead, we use a priority queue instead of being contracted.
Using CHs

Planning with CHs

- Bidirectional search
  - Only allowed to expand nodes with higher priority
  - Search meets at most important node

CH Properties

- Pros:
  - Very fast, especially just for path length
  - Optimal
- Cons:
  - Harder to implement
  - Not very dynamic
Distances can be computed very quickly. If we instead look full in road networks. Longer paths are likely to have more contraction hierarchies and why they have been so success-

happens when we look at the longest 1% of all problems. But, a peculiar things is worse than a the higher memory approach. The performance of the CH path and not to refine it, as this can be done incrementally.

For both the sector abstraction and CH, we evaluate the cost of finding the initial path here, and then measure the refine-

ment cost separately. The A* performance is based on a abstraction, so using a high-level

ment in node expansions, even on the hardest problems. Thus, the other implementation uses too much memory. We compare

in the best case, provid

abstraction on average,

24

×

Sector Abs (a)

25.3 43.2

250.5 302.3

24

×

Sector Abs (b)

137.8 150.4

70.3 68.6

16

×

A*

77.2 209.0

8

×

24

×

32

×

32

×

16

×

Figure 8: Map (a) is easy for abstraction but hard for CHs.

Table 2: Results for path planning.

<table>
<thead>
<tr>
<th></th>
<th>Easy for Abstraction</th>
<th>Harder for CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map (a)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map (b)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refinement in CHs is just a matter of unpacking edges. We give a general range of values that the refinement cost might
easier than finding paths in the first place.

data serves to show that refining the abstract paths is much cheaper than in sector abstractions. It is interesting to note small for smaller sector sizes. We measured several re-

level path node that is refined. The best and worst columns

We report the total cost of refinement for the sector abstrac-

tion in node expansions and total suboptimality in Table 3.

As the cost of refinement is length-dependent, we report the hard 1%

long 1%