ST. FRANCIS XAVIER UNIVERSITY

CSCI-564
CONSTRAINT PROCESSING AND HEURISTIC SEARCH

LECTURE 8 - AUTOMATICALLY CREATED HEURISTICS (CONTINUED)

Dr. Jean-Alexis Delamer

Recap

- Without a heuristic we can only search blindly.
  - Highly computational, we need to check each node.
- The heuristic guide the search.
  - It’s used to reduce the time complexity of the search.

Is it always true? Why?

Valtorta's Theorem

- A heuristic search is only beneficial if the computational effort to compute h is less than the savings generated by using h.

- Example:
  - Blind search, BFS $O(|V| + |E|)$
  - A* $O(|E|)$
    - Computing h, $O(|V| + |E|)$
    - Total complexity $O(|V| + 2|E|)$

~~You gain $O(|V|)$, good!~~

You added $O(|E|)$

Valtorta's Theorem

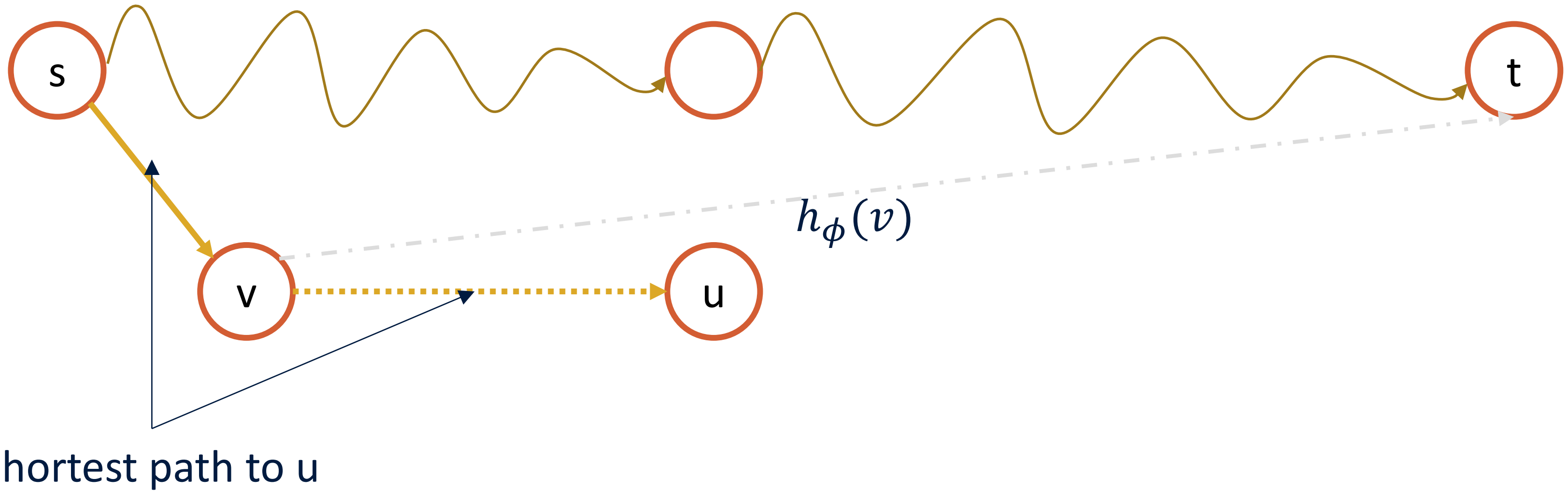
- Theorem (Valtorta's Theorem):
  - Let u be any state necessarily expanded, when the problem (s, t) is solved in S with BFS;
  - $\phi: S \rightarrow S'$ be any abstraction mapping, and heuristic $h(u)$ be computed by blindly searching from $\phi(u)$ to $\phi(t)$.
  - If the problem is solved by the A* algorithm using h, either u itself will be expanded, or $\phi(u)$ will be expanded.

Valtorta's Theorem

- Proof: When A* terminates, u will either be *closed*, *open*, or *unvisited*.
  - If u is *closed*, it has already been expanded.
  - If u is *open,* then $h_\phi(u)$ must have been computed during search. $h_\phi(u)$ is computed by searching S' starting at $\phi(u)$. If $\phi(u) \neq \phi(t)$, the first step is to expand $\phi(u)$; otherwise $h_\phi(u) = 0$ and u itself is expanded.
  - If u is *unvisited,* on every path from s to u there must be a state that was added to *open* during search but never expanded.

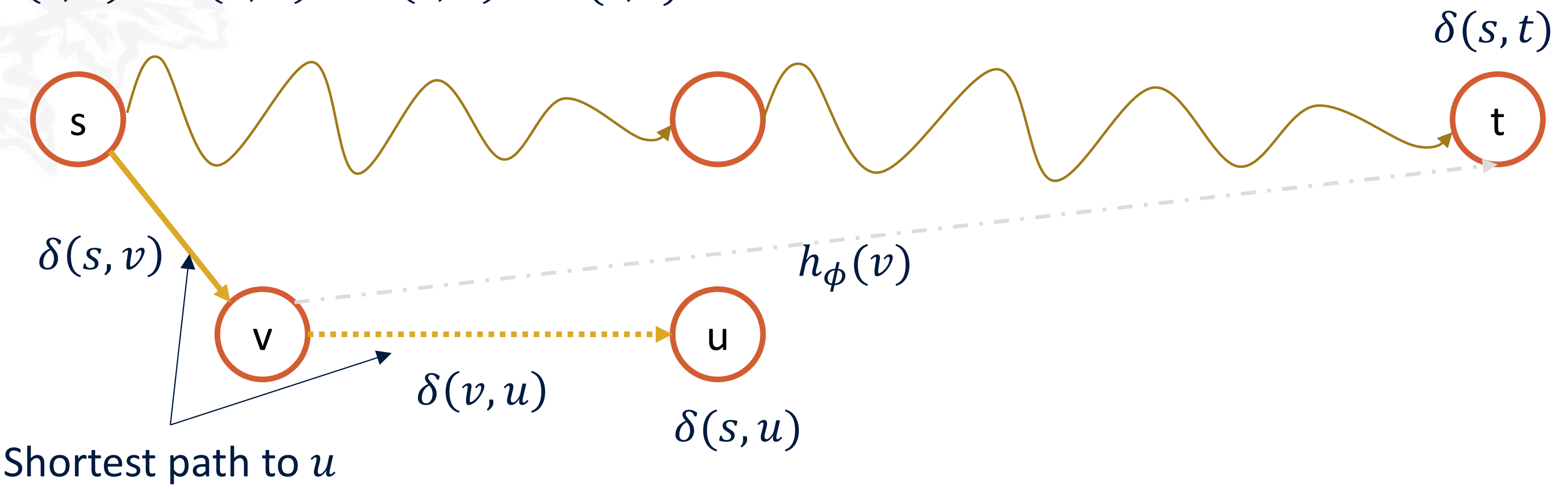
Valtorta's Theorem

- Proof: When A* terminates, u will either be *closed*, *open*, or *unvisited*.
  - If u is *unvisited,* on every path from s to u there must be a state that was added to *open* during search but never expanded.
    - Let v be any such state on the shortest path from s to u. Because v was opened, $h_\phi(v)$ must have been computed. We will show that computing $h_\phi(v), \phi(u)$ is necessarily expanded.



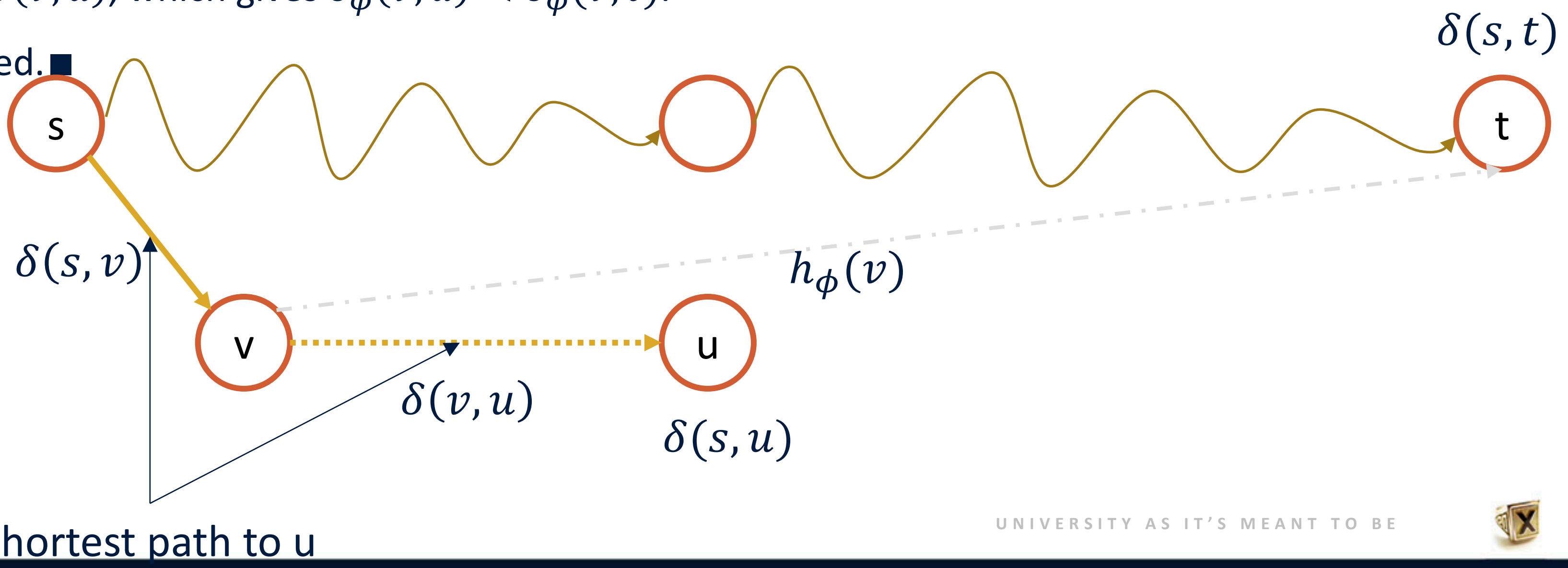
Valtorta's Theorem

- Proof: When A* terminates, u will either be *closed*, *open*, or *unvisited*.
  - If u is *unvisited,* on every path from s to u there must be a state that was added to *open* during search but never expanded.
    - Let v be any such state on the shortest path from s to u. Because v was opened, $h_\phi(v)$ must have been computed. To compute $h_\phi(v)$, $\phi(u)$ is necessarily expanded.
    - Because u is expanded by blind search, $\delta(s,u) < \delta(s,t)$. Because v is on the shortest path, $\delta(s,v) + \delta(v,u) = \delta(s,u) < \delta(s,t)$



Valtorta's Theorem

- Let v be any such state on the shortest path from s to u. Because v was opened, $h_\phi(v)$ must have been computed. To compute $h_\phi(v), \phi(u)$ is necessarily expanded.
- Because u is expanded by blind search, $\delta(s,u) < \delta(s,t)$. Because v is on the shortest path, $\delta(s,v) + \delta(v,u) = \delta(s,u) < \delta(s,t)$
- Because v was never expanded by A*, $\delta(s,v) + h_\phi(v) > \delta(s,t)$.
- Combining, $\delta(v,u) < h_\phi(v) = \delta(v,t)$
- Since ϕ is an abstraction, $\delta_\phi(v,u) < \delta(v,u)$, which gives $\delta_\phi(v,u) < \delta_\phi(v,t)$.
- Therefore, $\phi(u)$ is necessarily expanded. ■

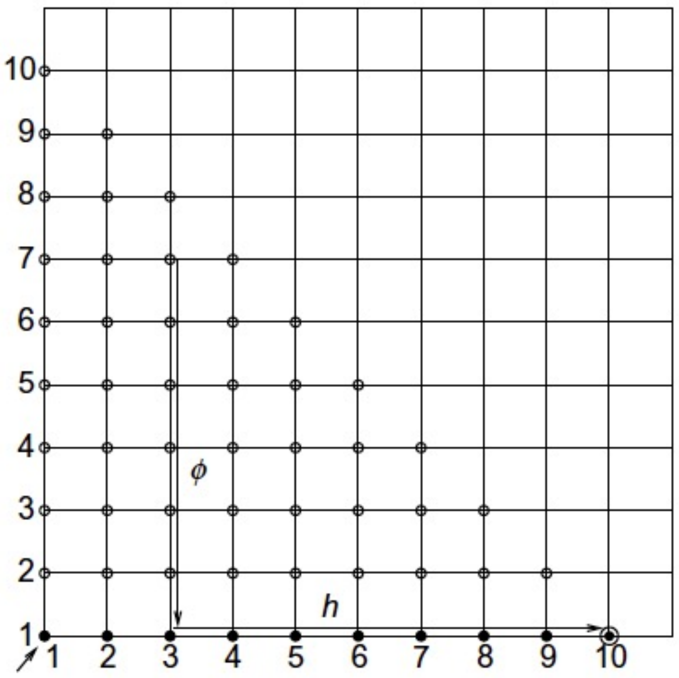


Valtorta's Theorem

- Corollary:
  - For an embedding ϕ, A* — using h computed by blind search in the abstract problem space — necessarily expands every state that is expanded by blind search in the original space.
- It assumes that the heuristic is calculated once for a problem instance.
  - You could amortize the cost , if you store the heuristic to reuse it.
- This theorem does not apply to homomorphism abstractions.

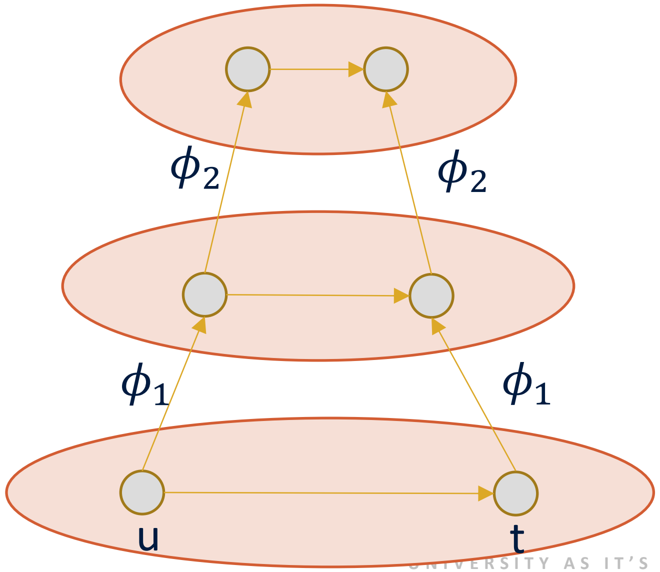
Valtorta's Theorem

- This theorem does not apply for homomorphism abstractions.
- Example:
  - Problem of finding a path between (1,1) and (1,N)
  - Abstraction transformation ignoring the second coordinate.
  - Uninformed search will expand $\Omega(N^2)$ nodes
  - Heuristic will only require $O(N)$.



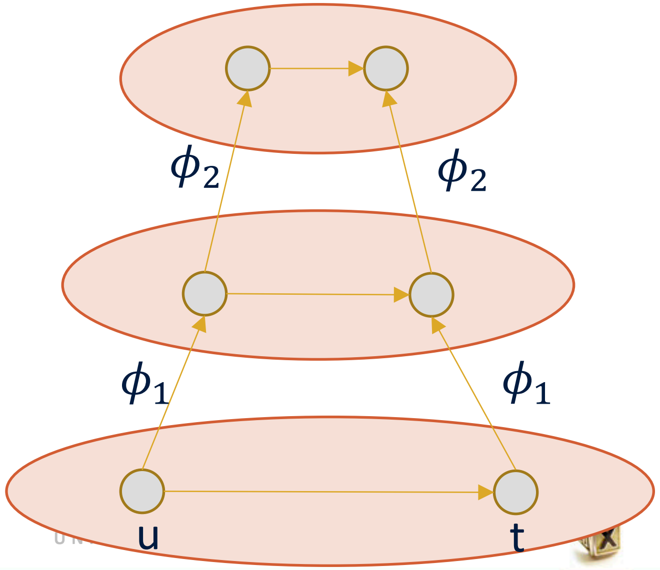
Hierarchical A*

- Hierarchical A*:
  - Use an arbitrary number of abstraction transformation layers.
  - Each layer named $\phi_1, \ldots, \phi_N$
  - When the heuristic call for the value u in the concrete problem, $\phi_1(u)$ is called.
  - Each layer calling the upper layer.



Hierarchical A*

- An issue is that it would repeatedly solve the same instances at the higher levels.
  - Because different concrete states can have the same state at higher levels.
- How can we solve this issue?
  - Save the heuristic values of all the nodes in shortest path computed at the abstract level.
- Does it respect the properties defined at the beginning?
  - No, the heuristic would no longer be monotone.



Hierarchical A*

- Definition (Monotonic Heuristic):
  - Let $(s_0, \dots, s_k)$ be any path, $g(s_i)$ be the path cost of $(s_0, \dots, s_k)$, and define $f(s_i) = g(s_i) + h(s_i)$. A goal estimate h is a monotone heuristic if $f(s_i) \leq f(s_j)$ for all $j > i, 0 \leq \mathrm{i}, \mathrm{j} \leq \mathrm{k}$; that is, the estimate of the total path cost is nondecreasing from a node to its successor
- The heuristic is nonmonotone in this case because:
  - Nodes that lay on the solution path of a previous search can have high h-values.
  - Whereas their neighbors off this path still have their original heuristic value
- You didn't explore everything yet!

Hierarchical A*

- What happens with nonmonotone heuristic?
  - Reopening of nodes.
  - Nodes can be closed even if the shortest path has not been found.

- A solution?
  - Yes, we don't care in this case.

Hierarchical A*

- Consider the following:
  - A node u can be prematurely closed if every shortest path passes through some nodes v for which the shortest path is known.
  - If no node v is part of the shortest path between s and t neither is u and the premature closing is irrelevant.
  - On the other hand, all nodes on the shortest path from v to t have already saved the exact estimate and will only be expanded once.

Hierarchical A*

- Optimal path caching:
  - An optimization technique
  - Save the value of $h^*(u) = \delta(u, T)$ and the exact solution path found
  - When a node u with $h^*(u)$ is encountered, the goal state is added to Open instead of expanding u.

Hierarchical A*

- What happen when you increase the number of layer?
  - More concrete states are assigned to the same abstract state.
  - The heuristic becomes less informative
    - Less discriminating.
- It’s called the granularity of abstractions

Exercise

- Represent Tower of Hanoi problem so it can be solved as a Hierarchical A*.