Abstraction in Pathfinding

with a focus on commercial video games

Nathan Sturtevant

UNIVERSITY OF ALBERTA

INFORMATICS CORE CIRCLE OF RESEARCH EXCELLENCE

Bridging contexts…

- Sven discussed many techniques for enhancing A* search
- There is another dimension along which we can optimize the performance of pathfinding algorithms

Abstraction

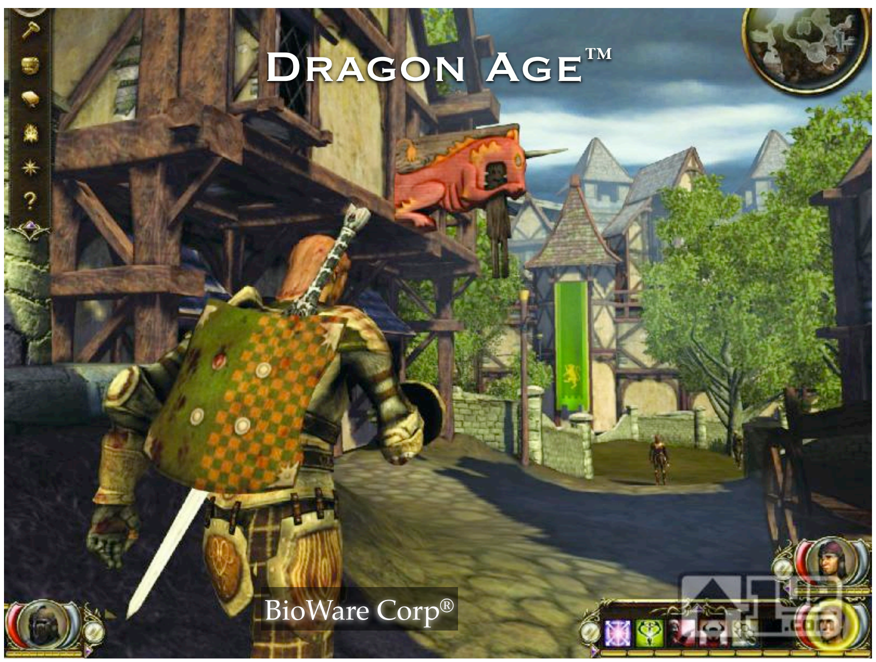
- Most search problems can be represented by a graph
- Build a smaller graph which retains most relevant information in the original graph
  - Similar to a low-resolution image
  - (Holte 96; Bulitko et. al. 07)

Why abstraction?

- Abstract graph is smaller
  - Search is cheaper
  - Defines subgoals in search
- Can be used for optimal or suboptimal solutions
  - Pattern databases

How is abstraction used?

- Almost all commercial video games use some type of abstraction
  - Unreal engine has Kynapse A.I. plug-in
  - Automatically builds high-level graph
  - Units can only walk on the graph
- Will describe the system built for BioWare Corp for their upcoming title Dragon Age (Sturtevant, 2007, AIIDE)



Motivation

- Games have tight memory budgets
  - ~5MB total memory for map data
  - 1024x1024 or larger maps
    - 1MB per byte per grid cell
- Can we use build an abstraction which minimizes memory usage?
  - Total memory usage by abstraction
  - Memory used during planning

Motivation

- Don't computers have lots of memory now?
  - Develop / design for low end
  - Models / graphics expensive
  - New gaming platforms
    - Nintendo DS
    - iPhone

Motivation

- Need to speed up search
  - Previous pathfinding engine was taking up to 100ms to plan
  - Ideally should plan in 1ms or less
    - 3-5ms for all planning per frame
  - May need to handle many units in the same time frame

Assumptions

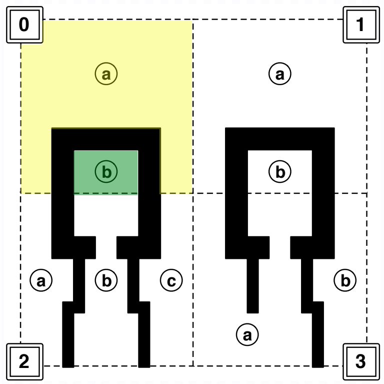
- Grid world
  - No true 3-d movement
- Cells can be blocked/free/weighted
- May be height difference between cells
- Units can move across real-valued space

Solution

- Build abstract graph from low-level data
  - Divide world into sectors
    - Sub-divide into regions
  - Maintain connectivity information between sectors

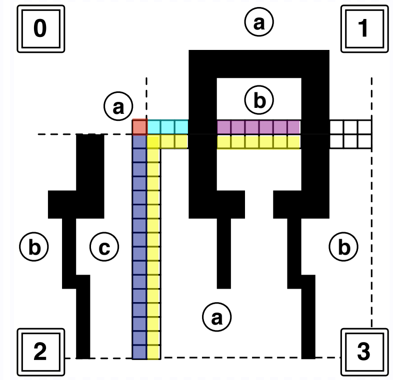
Sectors / Regions

- Divide world into large sectors
  - Fixed size
  - Index implicitly
- Divide sectors into regions
  - Regions entirely connected
  - Regions have a *center* point



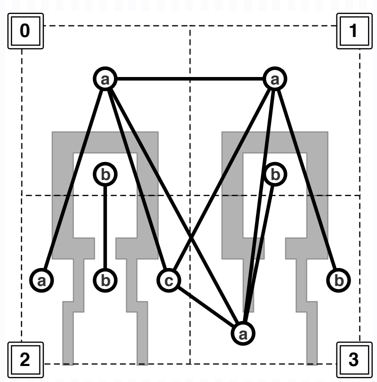
Edges

- Look at borders of regions to determine edges



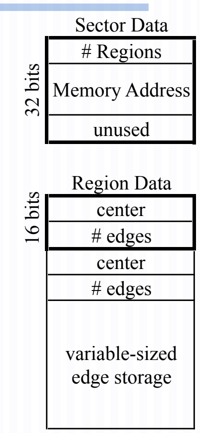
Abstract Graph

- Original Map:
  - 32x32 = 1024 cells
- Abstract Graph:
  - 9 nodes
  - 10 edges



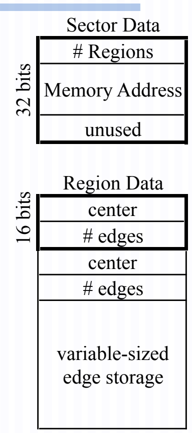
Memory Usage

- Sector data
  - Fixed size (32 bits)
- Region data
  - Variable sized
  - 8 bit region center
  - Edge count
  - 8 bits per edge



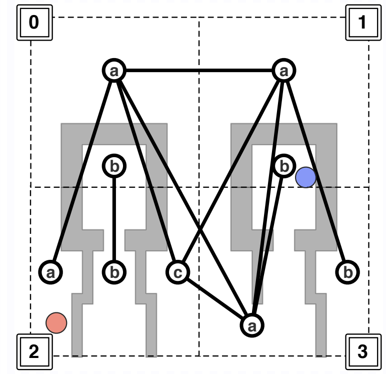
Memory Usage

- Can save more memory:
  - 16 bits for sectors
  - “Default” regions
  - Edges stored twice
  - Other optimizations



How is abstraction used?

- Need to find path between two points in the actual map
  - Find abstract region
  - Find abstract path
  - Refine



Find Abstract Region

- Begin with *x*/*y* location in real world
  - Sector implicit
- If sector only has 1 region, *done*
- Otherwise do BFS to find region center
  - Extra bits in grid can store region info
  - Pointers not needed

Find Abstract Path

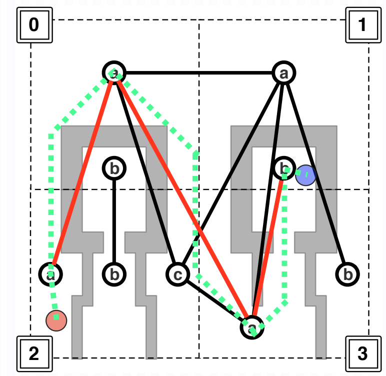
- Given sector/region for start and goal:
  - Use A* to find a complete abstract path
  - Use Manhattan/octile distance between region centers as both heuristic and edge cost

Refine

- Many different ways to use abstract path
- Simplest method:
  - Find path from start to first region
  - Find path to successive region centers
  - Find path from last region to goal

Usage Example

- Find abstract parents
- Find abstract path
- Find real path



Analysis

- Total pathfinding cost
- Optimizations
  - Region center placement
  - Reducing suboptimality
- Experimental verification

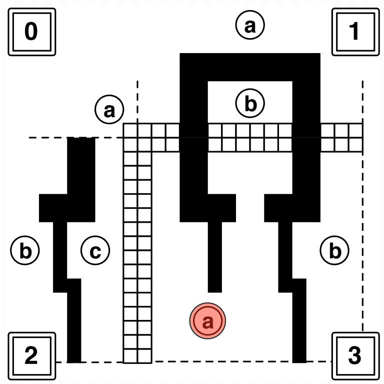
Total Pathfinding Cost

- Abstract planning
  - Depends on *path length*, *sector size*
- Refinement
  - Depends on *path length, edge refinement (region centers, suboptimality)*

- Maximize *sector size*

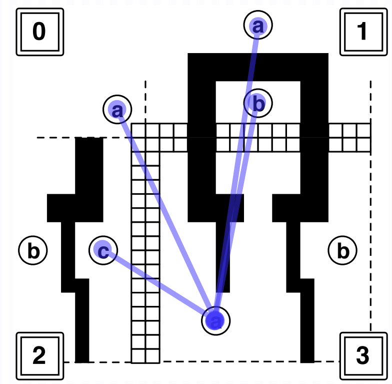
Optimizing Region Centers

- How to determine the region centers?
- Some locations are much better than others
- Harder with larger sector sizes



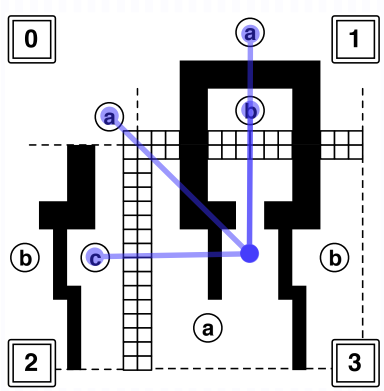
Optimizing Region Centers

- How to determine the region centers?
- Some locations are much better than others
- Harder with larger sector sizes



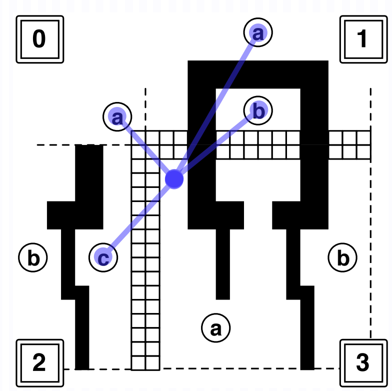
Optimizing Region Centers

- How to determine the region centers?
- Some locations are much better than others
- Harder with larger sector sizes



Optimizing Region Centers

- How to determine the region centers?
- Some locations are much better than others
- Harder with larger sector sizes

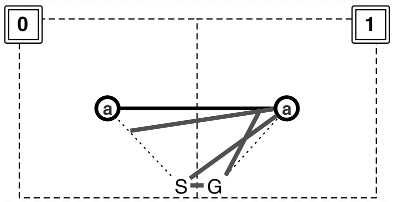


Optimizing Region Centers

- In a sector, for each cell in a region:
  - Measure A* cost to plan a path to each neighboring region from that cell
  - Choose the region center which minimizes the maximum cost
  - *Can optimize any cost function*

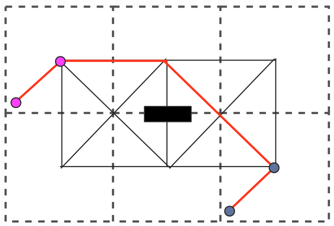
Pathfinding Optimization

- Refinement at start/goal can be inefficient
- Trim path segments
- Skip to next region at start/goal



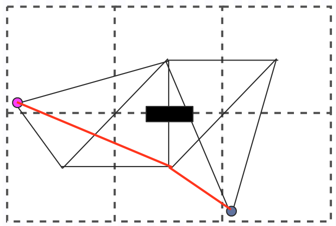
Sector-Related Errors

- All points within a sector/region are treated equally
- Adjust abstraction when performing search



Sector-Related Errors

- All points within a sector/region are treated equally
- Adjust abstraction when performing search



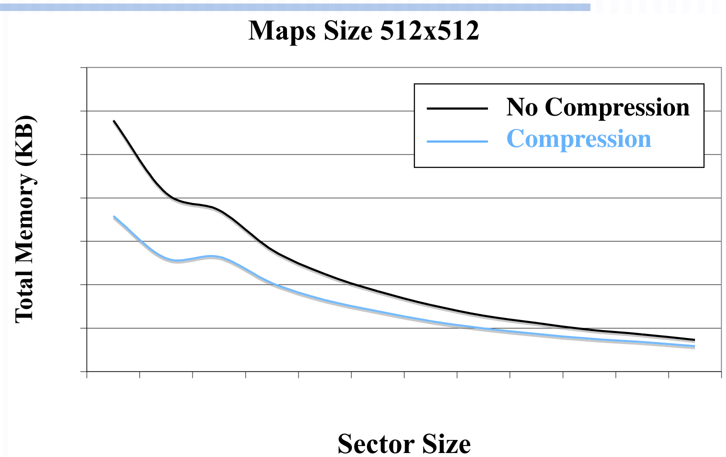
Experimental Results

- 93,000 paths over 120 maps
- Maps scaled to 512x512
- Paths length
  1…512
- Evaluate:
  - Memory
  - Region center optimization
  - Optimality
  - Total Work

Memory Usage

- How does the memory usage scale with sector size?
- How much memory can be saved with simple compression?
  - Don't store "default" sectors with
    1 region, 8 neighbors

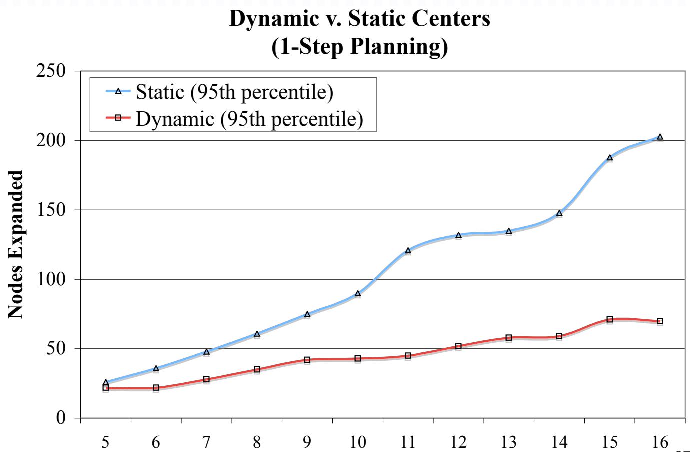
Memory Usage



Dynamic Region Centers

- Is there a gain to dynamically optimizing region centers?
- Measure 95% work done in one-step path refinement

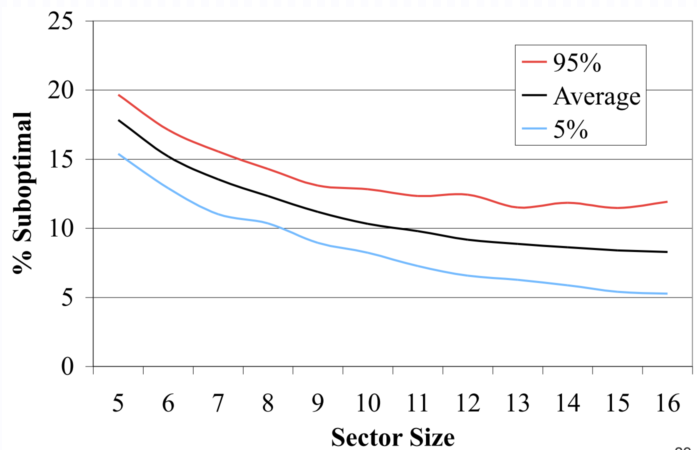
Dynamic Centers



Optimality

- Paths will not be optimal
  - Special cases for start/goal help
- Smoothing is applied as a post-processing step (not measured)

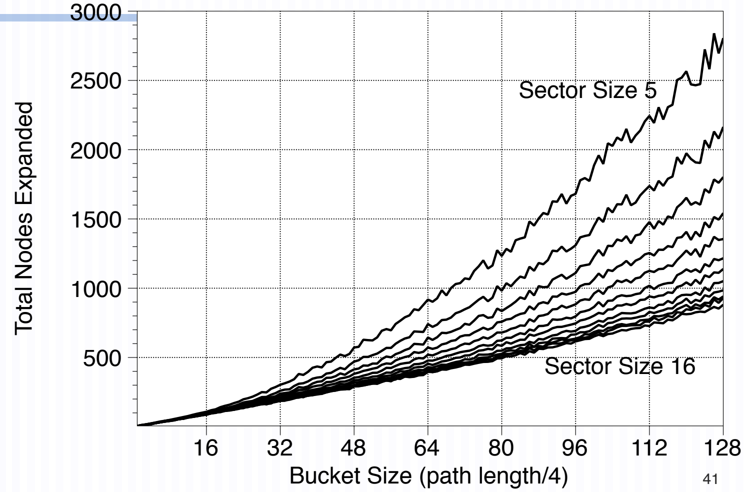
Optimality



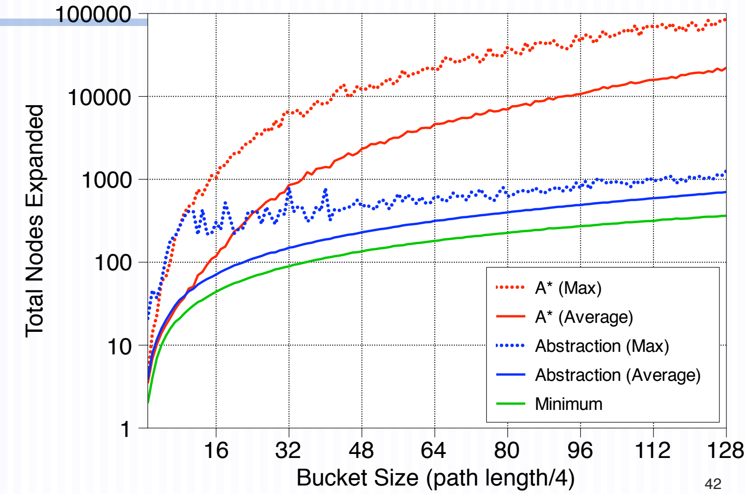
Total Work

- Compare total work by sector size
  - Find abstract path
  - Refine low-level path
- Compare to A*

Total Work

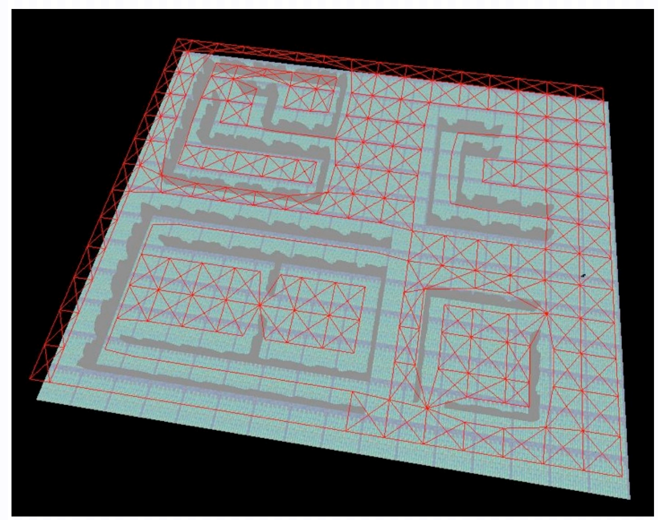


Total Work v. A*



Implementation

- Custom implementation for upcoming title Dragon Age™ (BioWare Corp®)
- Worked in-game during parts of 4 months
  - Initial implementation took two weeks
    - Rebuilt pathfinding core
  - Spent ~4 weeks optimizing code, adding smoothing, control structures, etc



Final Performance

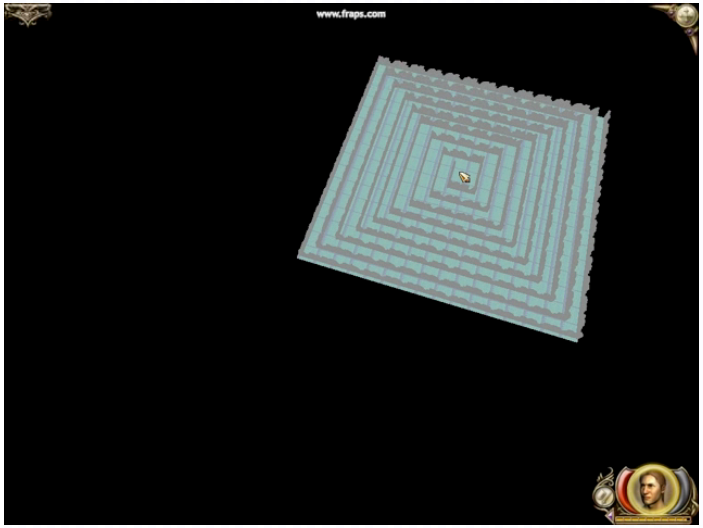
- About 0.1ms (100μs) per step of planning
  - Find abstract path
  - Refine 1 edge from abstract path
- Interleave planning and acting
  - Can plan for 30-50 units every frame
  - Units do not need to plan every frame
- Can “gracefully” degrade performance
  - Units offscreen don’t need to smooth

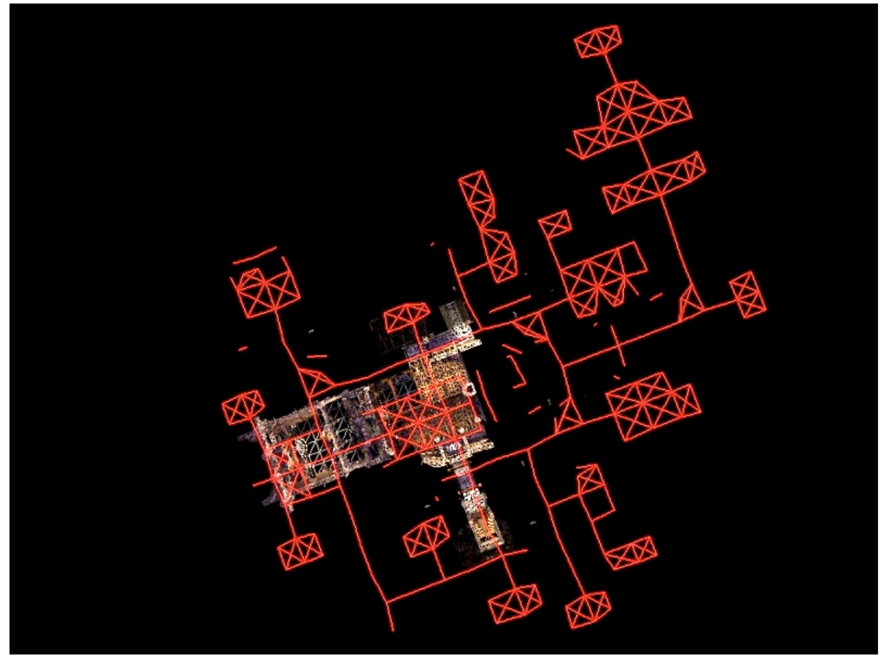
Memory Usage

- Memory usage is well within requirements
- Very little memory needed on a per-unit basis for planning
  - Abstract path
  - Current path
  - State of planning/smoothing

Data Structures

- A* uses:
  - Open list -- usually a heap
  - Closed list -- hash table
  - Back pointers -- reconstruct path
- Can’t store these on the map
- Simple implementation occasionally slow
- Allocate small closed list for each sector
  - Can quickly be cleared; no deallocation





Summary

- Units walk on real-space
- Abstract into a high-resolution 2-d grid
- Abstract again into coarse graph

- Units pretend to live on high-resolution grid
- Michael will talk about getting rid of the 2-d grid

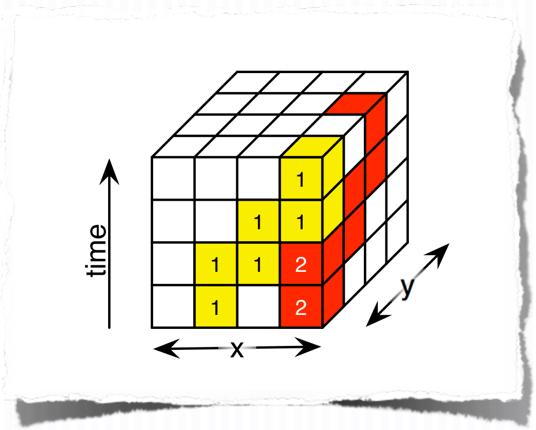
That's great but…

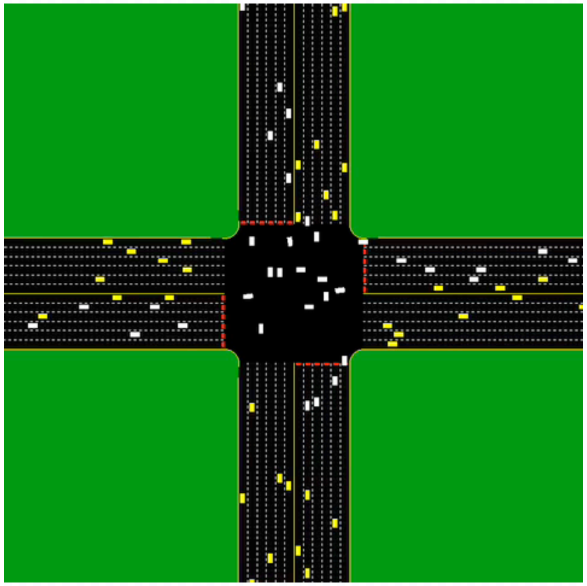
- In many domains, pathfinding involves multiple units
- How can units cooperate when planning?
  - Ignore each other and replan
  - Using 'flocking' methods to avoid other units
  - Explicitly cooperate
    - (Dresner and Stone, 2008, JAIR)
    - (Silver, 2005, AIIDE)
    - (Sturtevant and Buro, 2006, AIIDE)

Dresner & Stone

- Traffic management problem
  - Can cooperative cars increase traffic throughput?
- Centralized system manages reservations
  - Can a car get through the intersection safely?
  - Tries several different speeds
  - Forces cars to wait until the can get a reservation

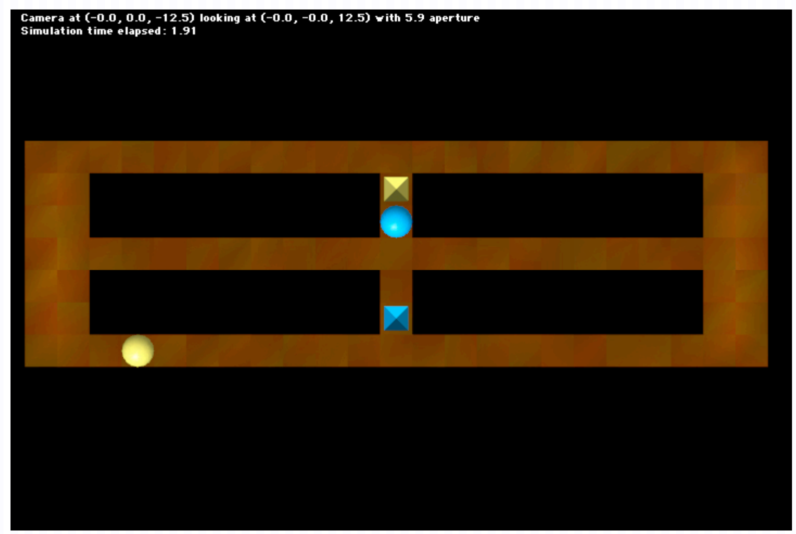
Data Structure

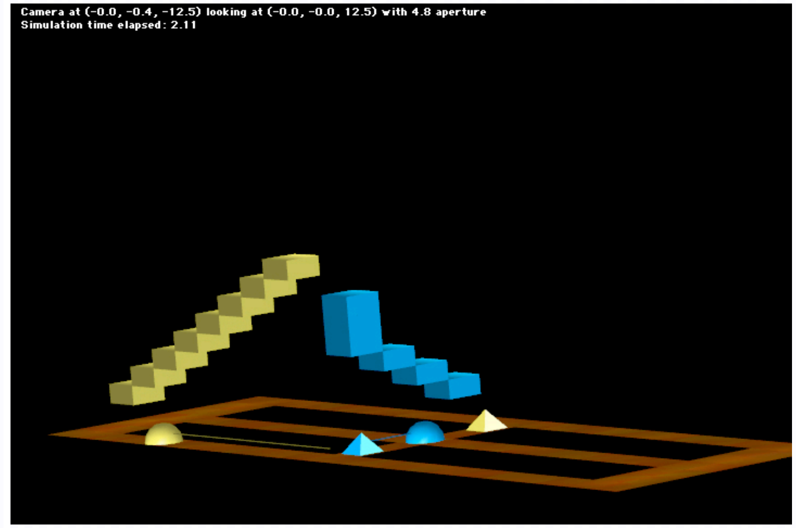


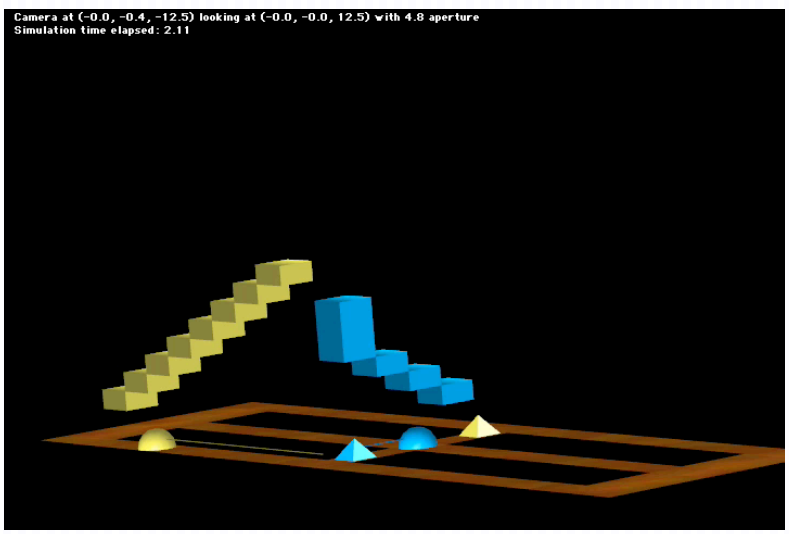


Generalized Cooperative Pathfinding

- Goal: Multiple agents cooperate during path planning and execution
  - Generalized travel (eg no lanes)
  - Centralized reservation system
  - Use abstraction to reduce costs







Possible Strategies

- Plan all units simultaneously
  - Computationally intractable
  - $(units^{actions})^{depth}$
- Plan individual units
  - Not complete
  - A lot of techniques needed to be practical

Overview

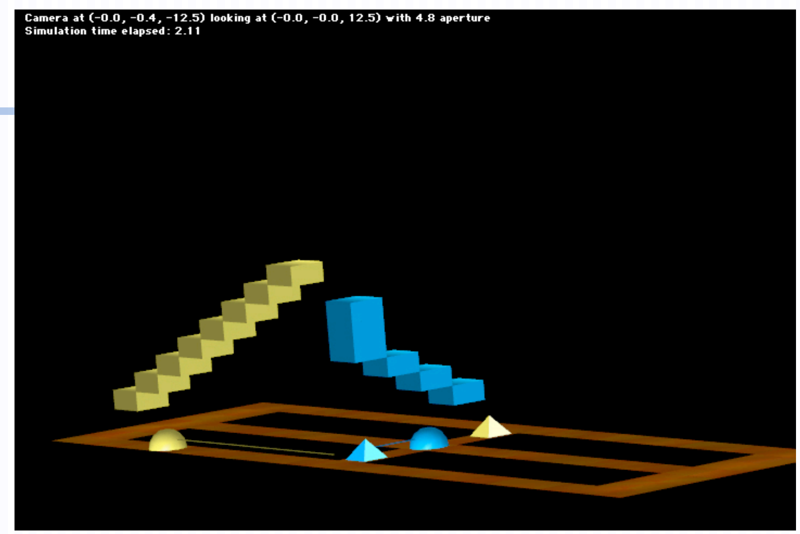
- Why problem is hard
- What techniques simplify the problem
- Improving performance with abstraction
- Evaluation

WHCA*(*w*)

- Windowed Hierarchical Cooperative A*
  - Cooperative A*
  - Hierarchical Heuristic
  - Windowed cooperation
- Silver, 2005

WHCA*

- Use a hash table to store time-space indexed reservations
  - Constant time acces
  - Is a space/time cell free?
  - Reserve a space/time cell
  - Free a space/time cell

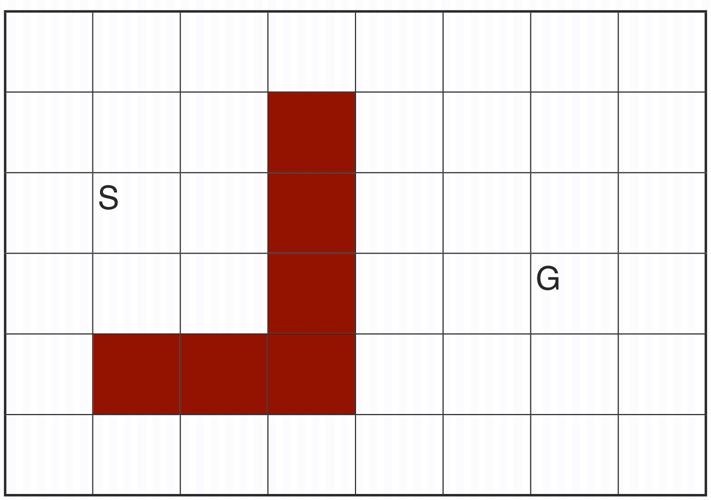


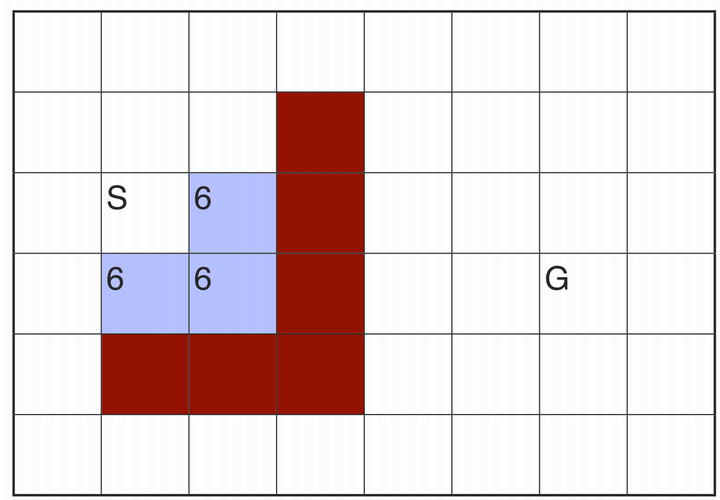
A*

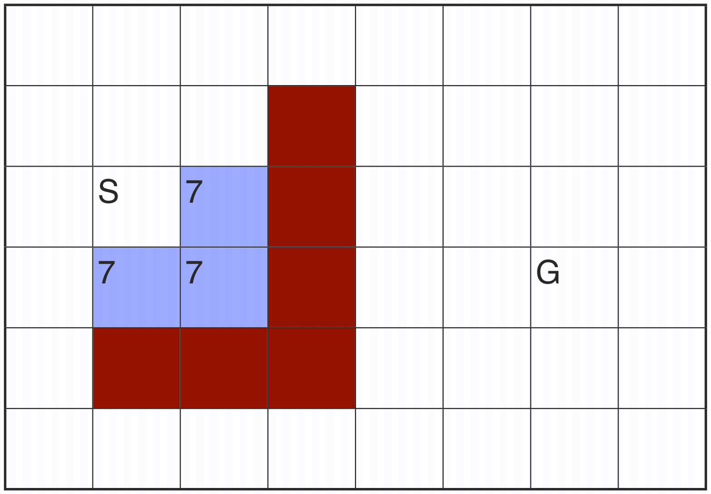
- A* relies on a heuristic to guide search
  - Poor heuristics cause extra node expansions
  - Cost is the **area** in which the heuristic is poor

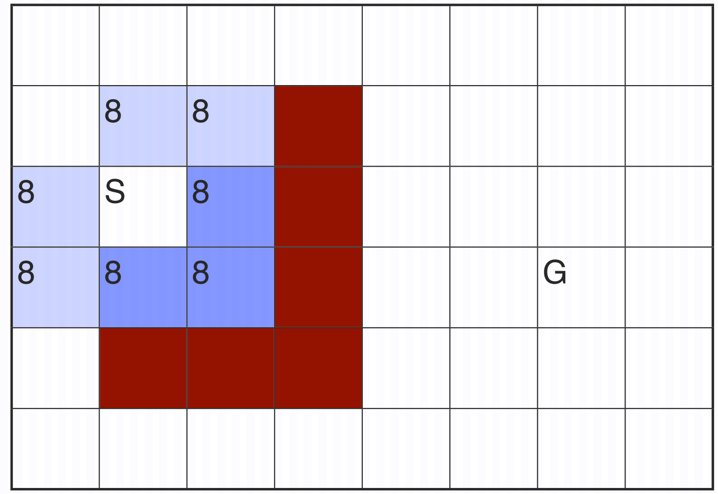
Cooperative A*

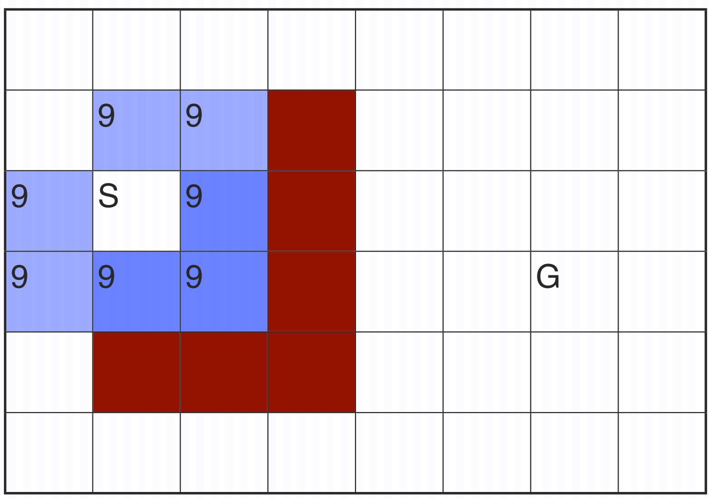
- 3-dimensional search problem
  - x-location, y-location, time
  - Still need a heuristic
    - Cost is the **area** in which the heuristic is poor *times* the time to get out of that area = **volume**

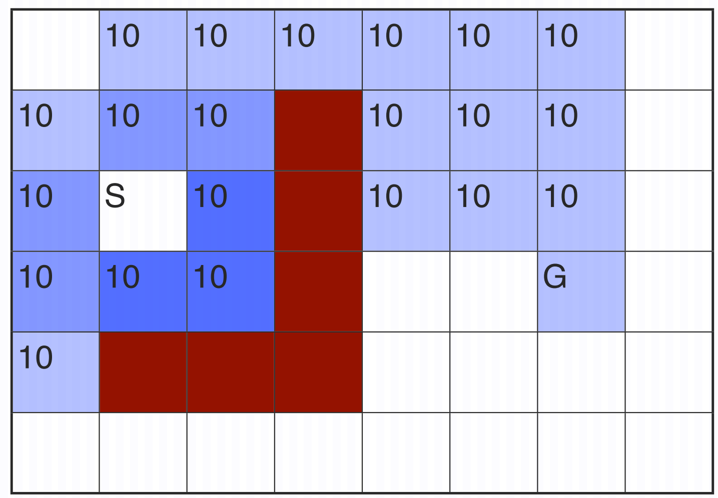





S
7
7
7
G


8
8
8
S
8
8
8
8
G


9
9
9
S
9
9
9
9
G


10
10
10
10
10
10
10
10
10
10
10
10
10
S
10
10
10
10
10
10
10
10
G
10

Heuristics

- Need a very accurate heuristic
- Where can we get a heuristic?
  - Run A* from the goal to the start state to get h() value for many states

	S						
						G	

	8+2	7+3	6+4	5+5	4+6	3+7	
	9+1	8+2		4+4	3+5	2+6	3+7
	S			3+3	2+4	1+5	2+6
				2+4	1+5	G	1+7
				3+5	2+6	1+7	2+8
g+h				4+6	3+7	2+8	

	8	7	6	5	4	3	
	9	8		4	3	2	3
	S			3	2	1	2
				2	1	G	1
				3	2	1	2
				4	3	2	

Windowed Search

- We now have a perfect heuristic
  - With a perfect heuristic only 1-step lookahead is needed
  - Stop search at any time and be guaranteed to be on a path to the goal
- Do *k*-step lookahead in cooperative space

WHCA*(*k*)

- Do single A* search from goal to start
  - Do *k*-step forward cooperative search
  - Expand original search if new heuristic values needed

WHCA* Drawbacks

- First step is expensive
  - Compute complete reverse A* search
  - Compute forward CA* search
- Memory per unit is expensive
  - Keep whole search frontier in memory
- Goal State can't change

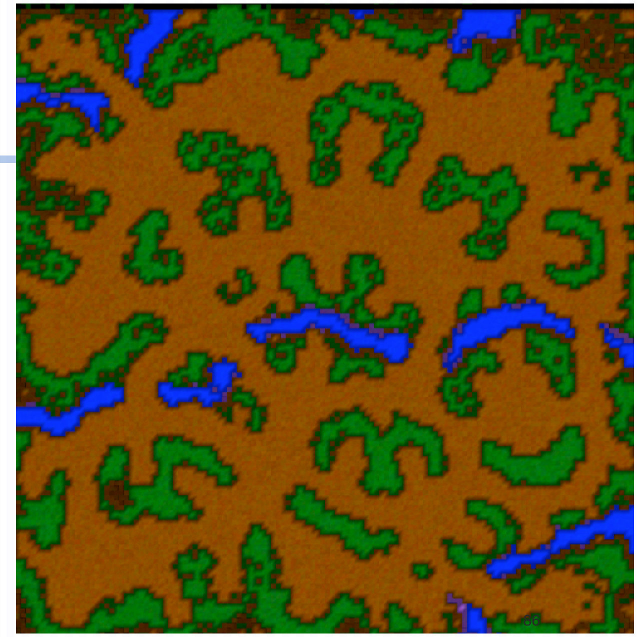
Improving WHCA*

- Abstraction
  - Widely used idea (eg Holte, 1996)
- Two possible usages
  - WHCA*(*w*, *a*)
  - CPRA*(*k*)

Abstraction

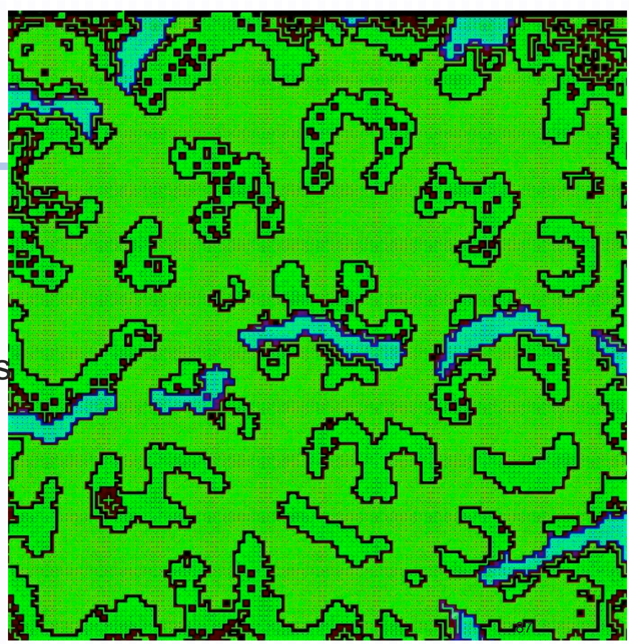
- Use fine-grained map abstraction
  - Dragon Age abstraction abstractions 16x16 sectors in one step
- Instead abstract 2x2 sectors in one step
  - Or: abstract small cliques (4 nodes) in the map
  - Theoretical work suggests this minimizes pathfinding computation
    - (Holte, 96; Sturtevant and Jansen, 07)

Sample Map



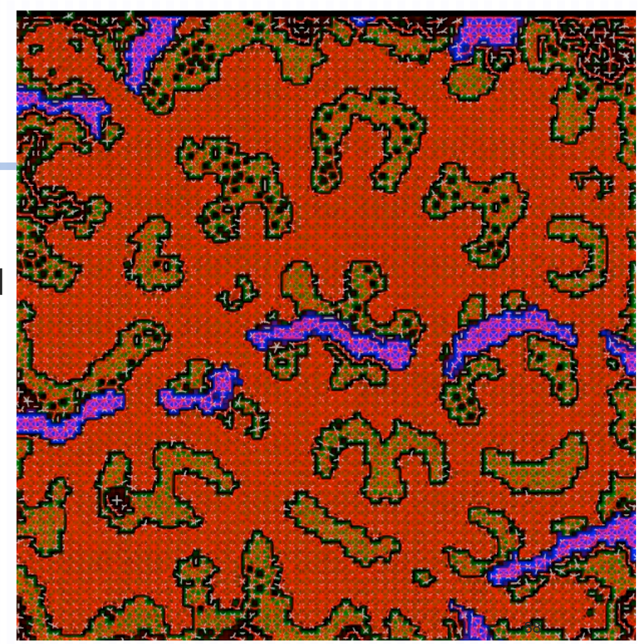
Base Graph

16,807 nodes



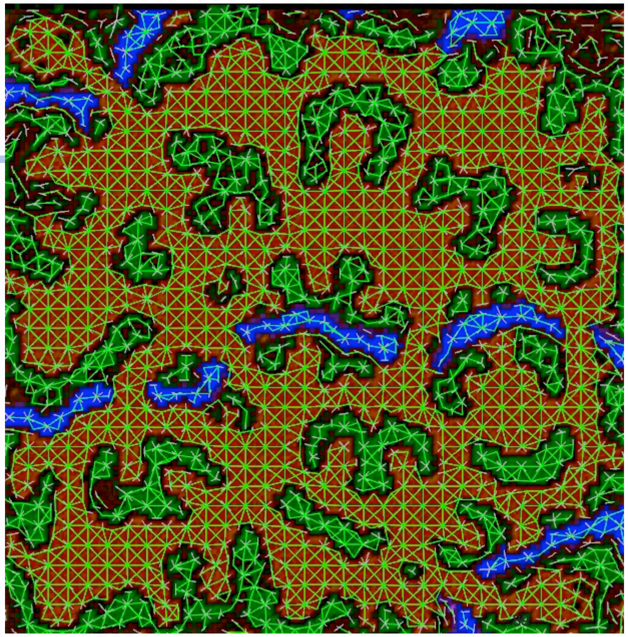
Abstraction 1

5,212 nodes



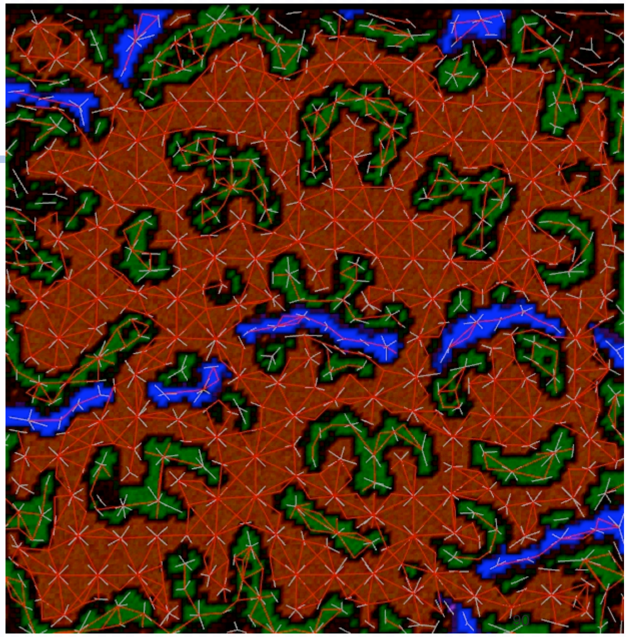
Abstraction 2

1,919 nodes



Abstraction 3

771 nodes

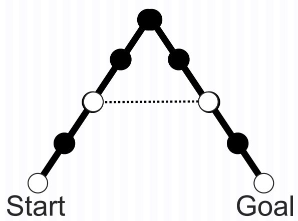


WHCA*(*k*, *a*)

- Same as WHCA*(*k*) but do reverse A* search at abstract level *a*
- Keep smaller A* open/closed list in memory
- Faster A* computation
- Eventually less accurate

PRA*

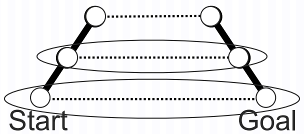
- Partial-Refinement A*
  - Use multiple abstraction levels
  - Refine abstract paths using A*



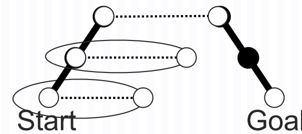
Pathfinding

- 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

PRA*(∞)



PRA*(*k*)

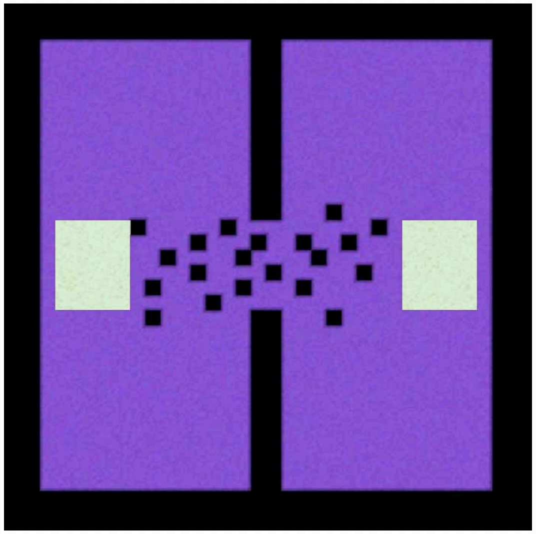


CPRA*(*k*)

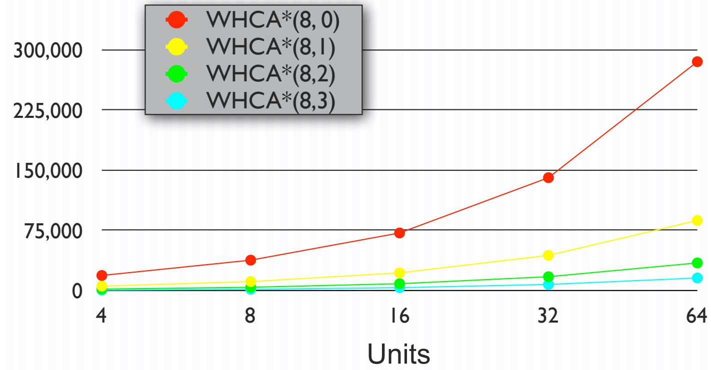
- Same as PRA*(*k*), but do WHCA*(*k*, 1) at last refinement level
- Only plan part of total path
  - Much lower first-step cost
- Repeated WHCA* calls after executing each path

Experiments

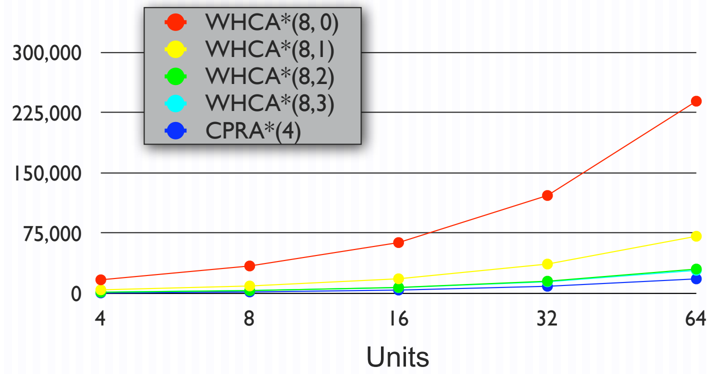
- Run algorithms on 256x256 map
  - Place units on opposite sides of map and ask them to cross sides
- Report 95%



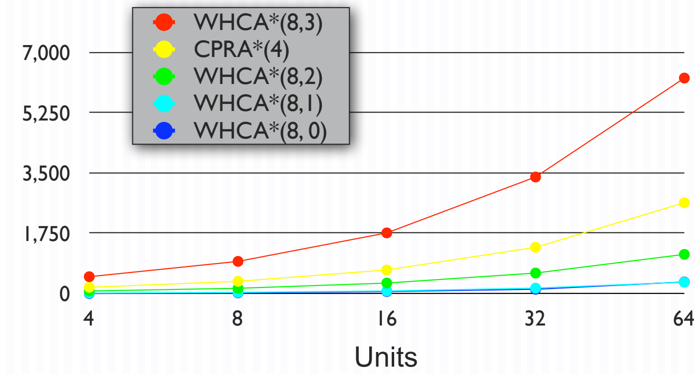
Memory Usage



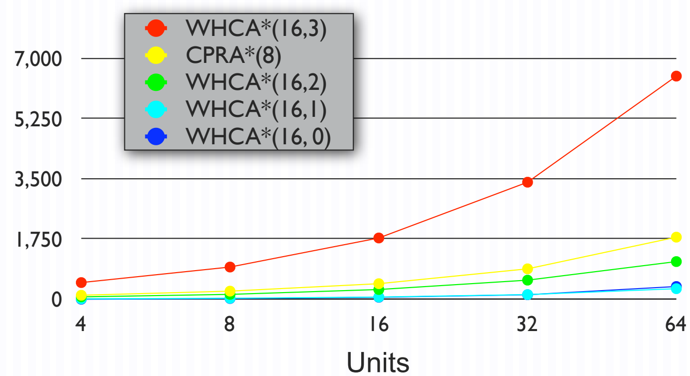
Nodes
First second



Nodes
Average per second



Nodes
Average per second

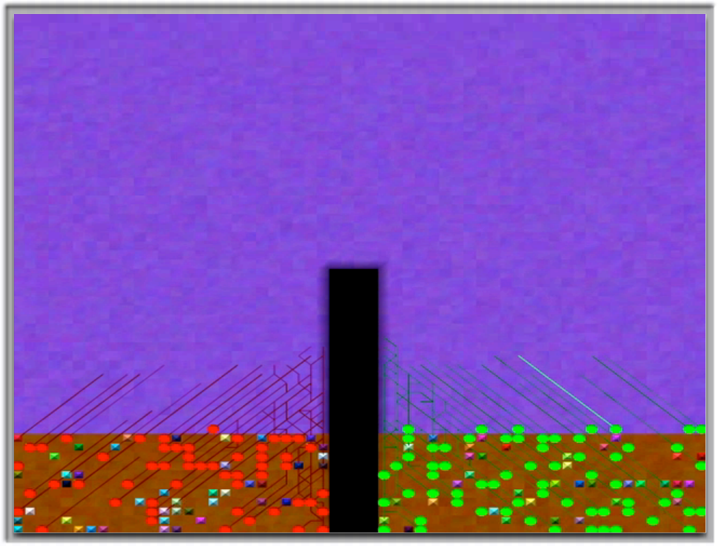


Generalizing

- General technique for *n*-dimensional pathfinding problems
  - Solve problem in *n*-1 dimensional space
  - Use as heuristic in *n*-dimensional search
  - If possible use "lower resolution" version of *n*-1 dimensional problem

But…

- How well does it work with lots of units in open space?
  - Not as well as one might expect

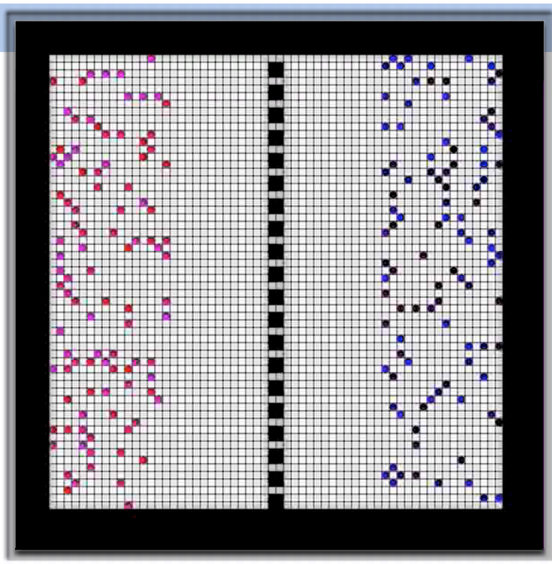


But…

- How well does it work with lots of units in open space?
  - Not as well as one might expect

- Units are searching for the *shortest* path
  - Prefer shorter paths over paths which have a higher probability of success

Real crowd movement



Simulated crowd movement

Some perspective

- Static 2-d search is cheaper than 3-d search
- Static information about other units isn't very useful

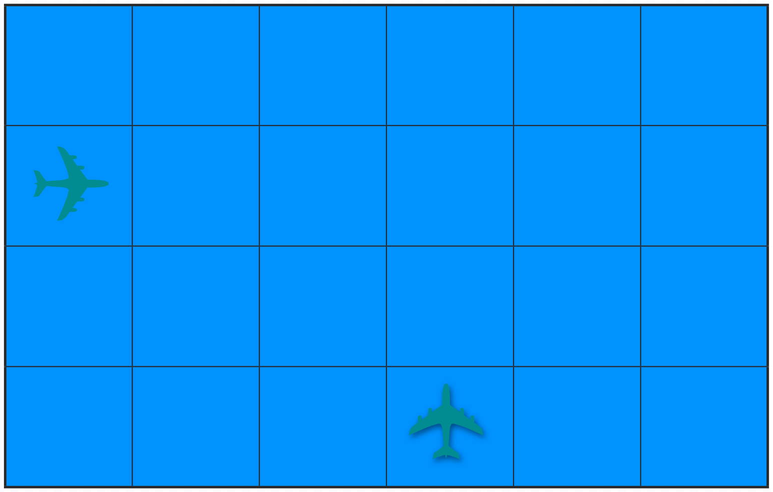
- Is there any other static information that we can retain?

Static information about motion.

Retained Information

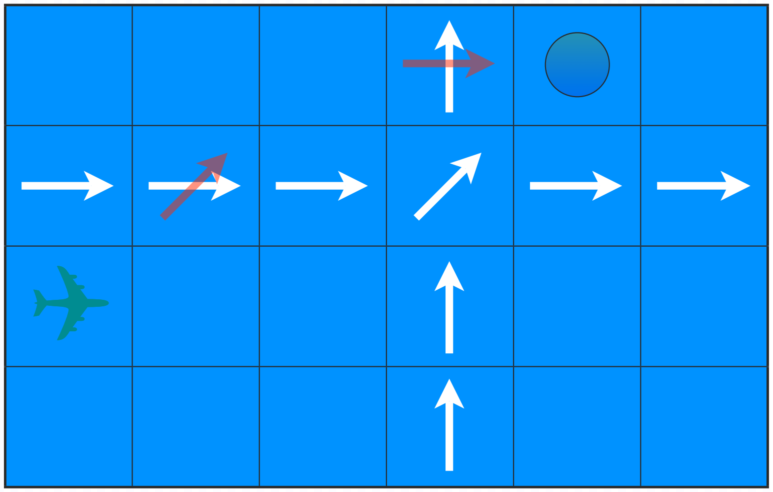
- Direction Vector
  - Associated with a location on a map
  - Which direction units travel through the location
  - Updated dynamically as units move
- Direction Map
  - Direction vectors for every location on the map

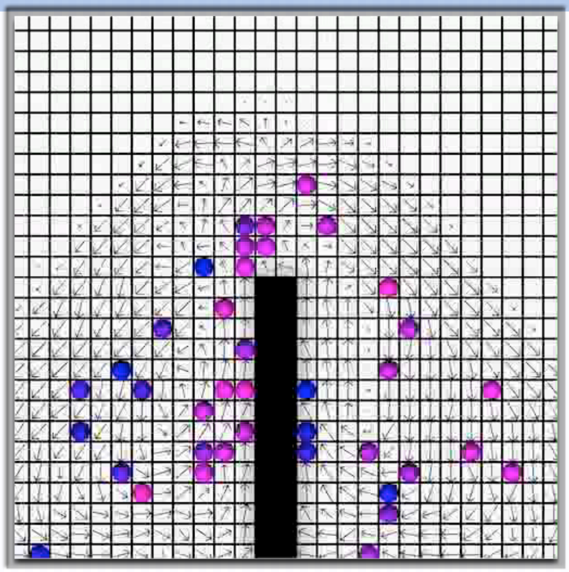
- Similar to flow fields used for flocking

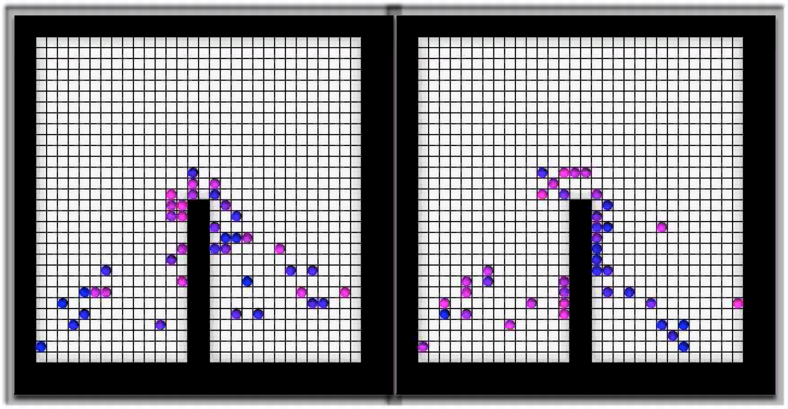


Now what…?

- What do we do with all these arrows?
- During planning:
  - Traveling in the same direction of an arrow is cheaper
  - Traveling in the opposite direction is more expensive

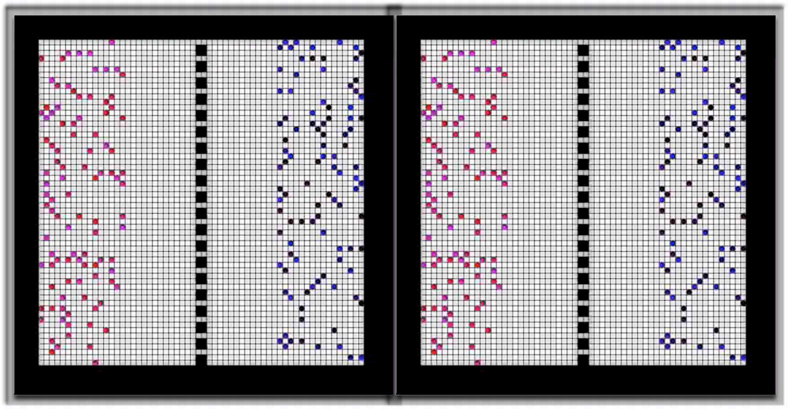






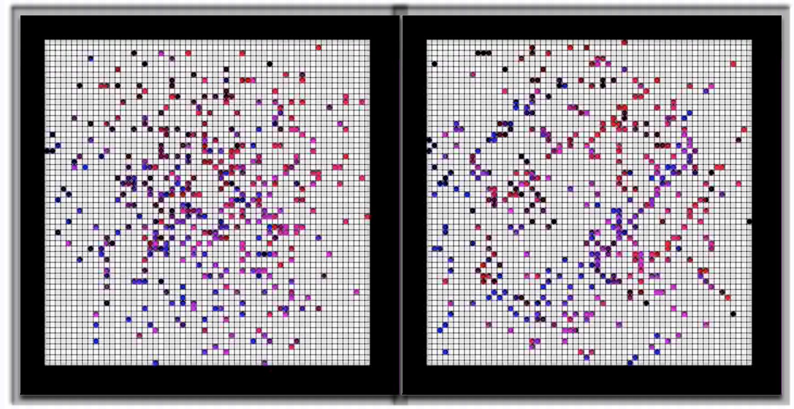
Uncoordinated Unit Behavior

Coordinated Unit Behavior



Uncoordinated Unit Behavior

Coordinated Unit Behavior



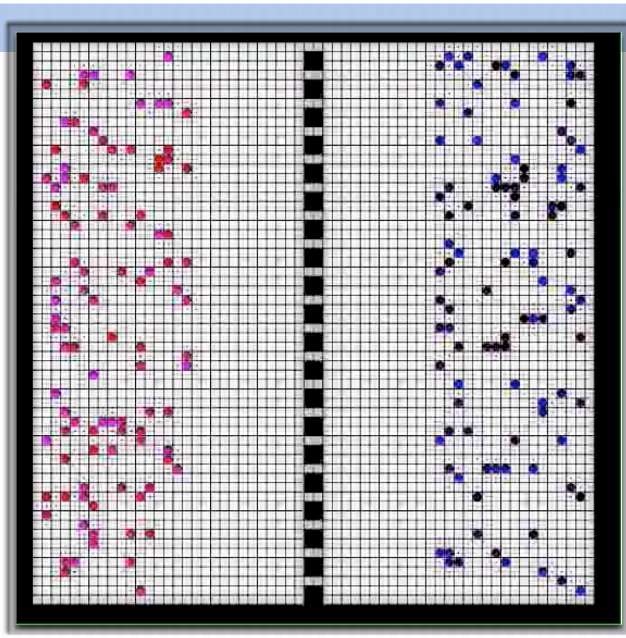
Uncoordinated Unit Behavior

Coordinated Unit Behavior

Practical Considerations

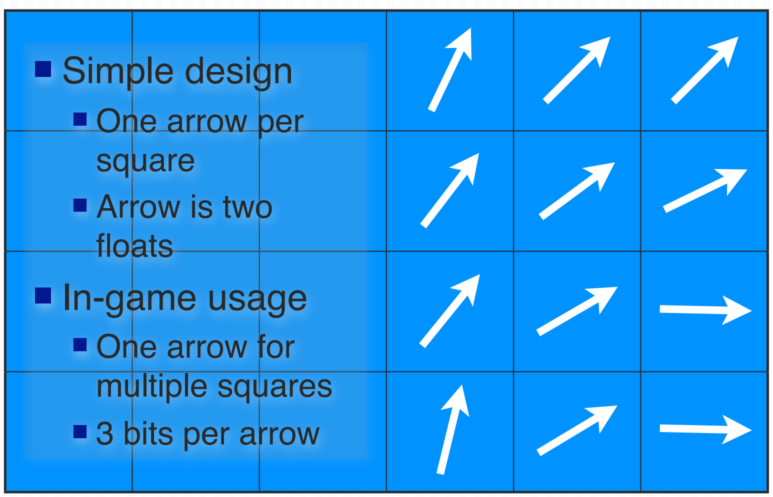
Other considerations

- Where do the initial weights come from?
- How much memory does it take to store the weights?
- What is the additional planning cost?



Other considerations

- Where do the initial weights come from?
- How much memory does it take to store the weights?
- What is the additional planning cost?



- Simple design
  - One arrow per square
  - Arrow is two floats
- In-game usage
  - One arrow for multiple squares
  - 3 bits per arrow

Other considerations

- Where do the initial weights come from?
- How much memory does it take to store the weights?
- What is the additional planning cost?

Planning Cost

- Planning using direction maps is more expensive
  - Weighted A* can reduce the cost
  - Use abstraction to reduce planning length
  - Can maintain direction maps for classes of units, or only in congested areas of the map

Summary

- Abstraction techniques very effective across a variety of problems in reducing planning costs
  - Used for defining subgoals in search
    - Dragon Age
  - Used for heuristics in search
    - Cooperative pathfinding
- Many different ways of applying abstraction
  - Best method depends on problem constraints

Summary

- Abstraction is orthogonal to many other search enhancements
  - Everything Sven talked about could be used on one or more levels of abstraction
  - Rich toolbox for balancing performance in any particular domain

Thanks!

- Comments, questions?

- Co-collaborators:
  - Markus Enzenberger
  - Renee Jansen
  - Michael Buro
  - Vadim Bulitko