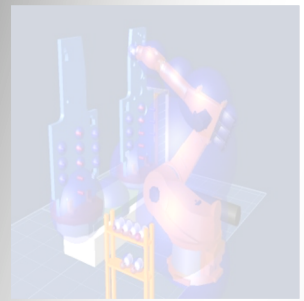
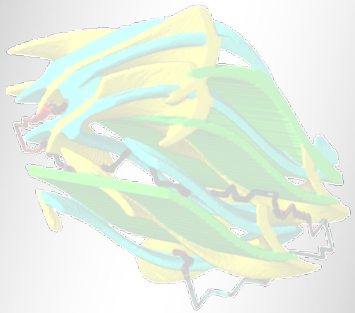
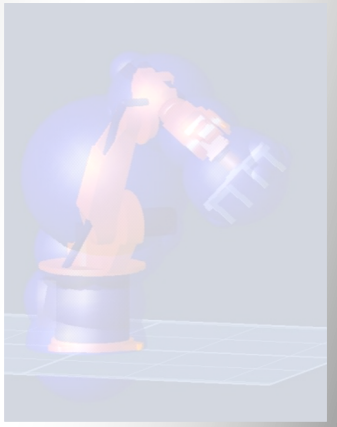
A* - Algorithm for path planning

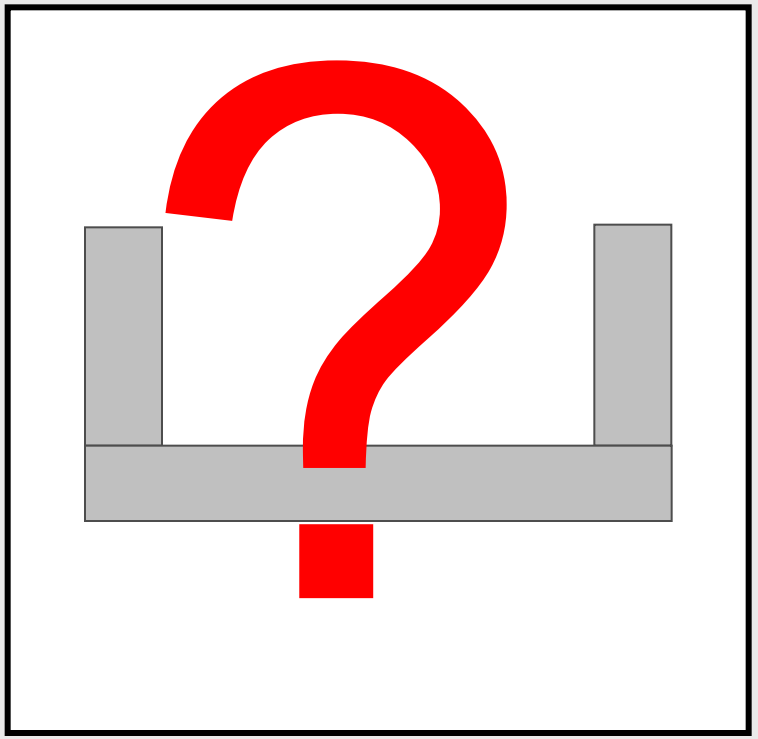






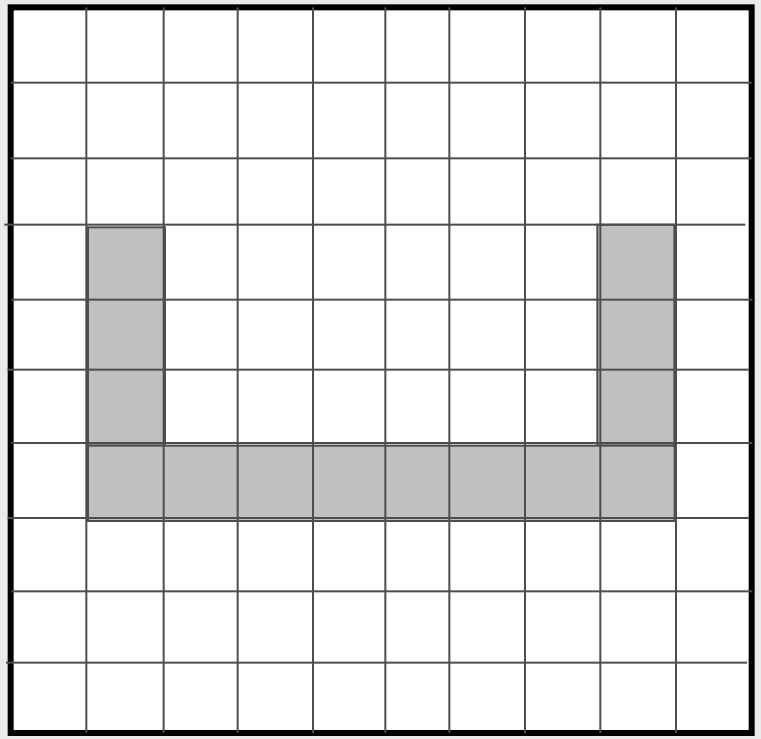
Discretisation of C-space

- To use an A*-algorithm as path planning algorithm, C-space must be discretisized.



Discretisation of C-space

- To use an A*-algorithm as path planning algorithm, C-space must be discretisized.
- After discretisation C-space can be interpreted as a grid on which A* can „take place“
- The joint values are trans-formed to **grid coordinates** based on joint limits and numbers of discretisation steps



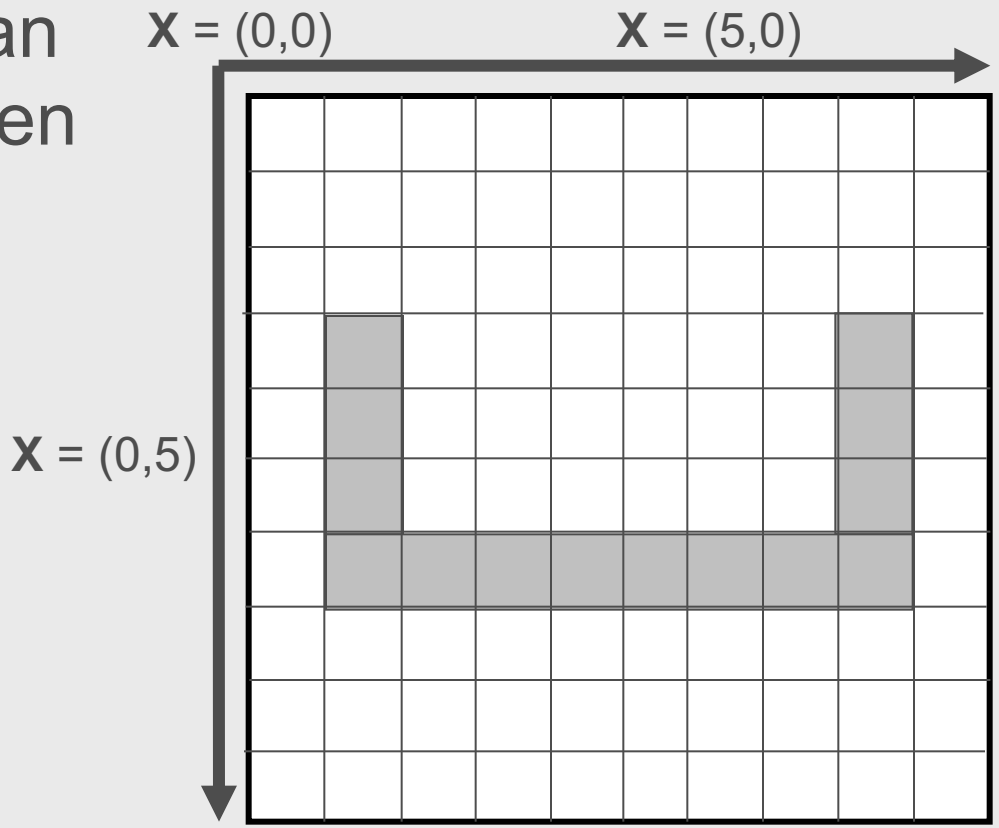
Discretisation of C-space: grid coordinates

- Every grid coordinate correspondents exactly to one configuration
- Let $\Delta\Theta_i$ the chosen discretisation and $\Theta_{\min,i}$ the lower limit of joint i, then φ_i can be determined by a given grid coordinate x_i:

$$\varphi_i = \Theta_{\min,i} + \Delta\Theta_i x_i$$

- Other way around:

$$x_i = \left\lfloor \frac{\varphi_i - \Theta_{\min,i}}{\Delta\Theta_i} + \frac{1}{2} \right\rfloor$$



A* algorithm in more detail

```
procedure A* (S;G)
tNode : S;                          /* INPUT: Start node */
tNode : G;                          /* INPUT: Goal node */
begin
    tNode :       n;                /* Temporary node */
    tNodeList :   s;                /* Successor node list */
    tHeap :       OPEN;             /* Nodes to be expanded */
    tHashtable :  CLOSED;           /* Nodes already expanded */
    tBoolean :    goalFound;        /* TRUE, if Goal is found */
    init (OPEN;CLOSED; S) ;
    goalFound = FALSE;
    while (OPEN ≠ ∅) do begin
         n = best (OPEN) ;
         if (n == G) then
                  goalFound = TRUE;
                  break;
         endif
         s = expandNode (n) ;
         handleSuccessors (OPEN;CLOSED; s) ;
    end
    if (goalFound == TRUE) then
         printSolution (: : : ) ;
    endif
    return (goalFound);
end;
```

A* algorithm in more detail

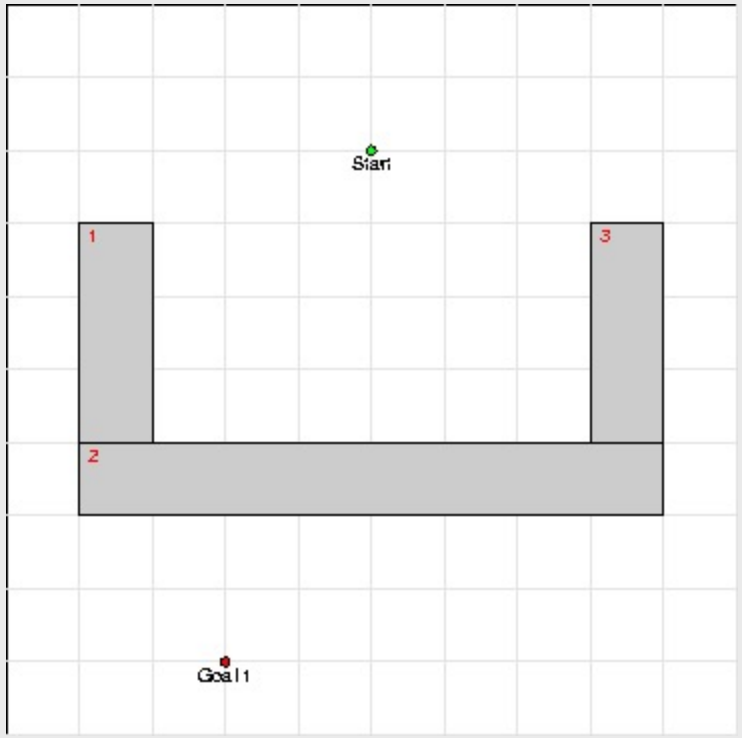
```
procedure bestfirst (S;G)
tNode : S;                          /* INPUT: Start node */
tNode : G;                          /* INPUT: Goal node */
begin
    tNode :       n;                /* Temporary node */
    tNodeList :   s;                /* Successor node list */
    tHeap :       OPEN;             /* Nodes to be expanded */
    tHashtable :  CLOSED;           /* Nodes already expanded */
    tBoolean :    goalFound;        /* TRUE, if Goal is found */
    init (OPEN;CLOSED; S) ;
    goalFound = FALSE;
    while (OPEN ≠ ∅) do begin
         n = best (OPEN) ;
         if (n == G) then
                  goalFound = TRUE;
                  break;
         endif
         s = expandNode (n) ;
         handleSuccessors (OPEN;CLOSED; s) ;
    end
    if (goalFound == TRUE) then
         printSolution (: : : ) ;
    endif
    return (goalFound);
end;
```

Place to modify algorithm

A*-search

- Graph search (A*) in implicit C-space

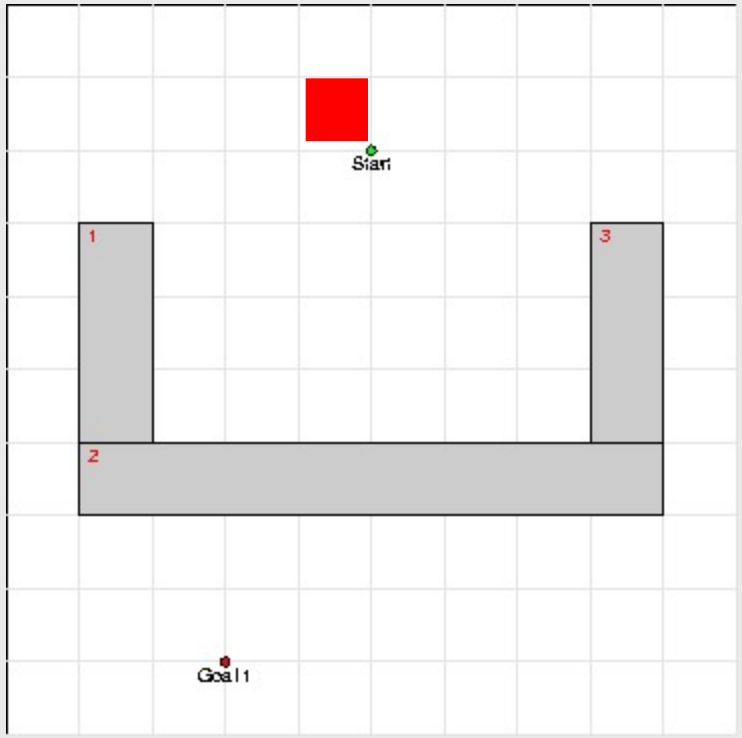
```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```



A*-search

- Graph search (A*) in implicit C-space

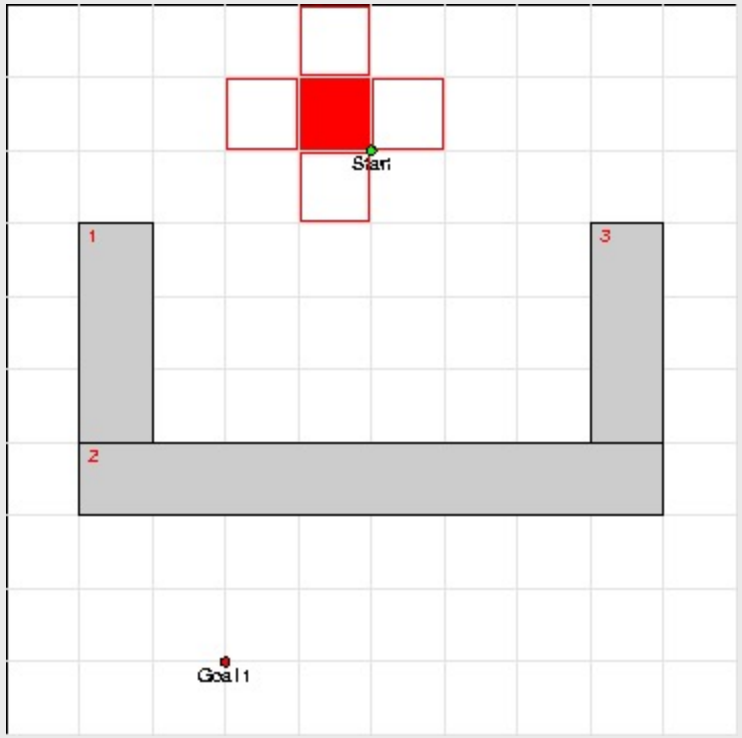
```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```



A*-search

- Graph search (A*) in implicit C-space

```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```

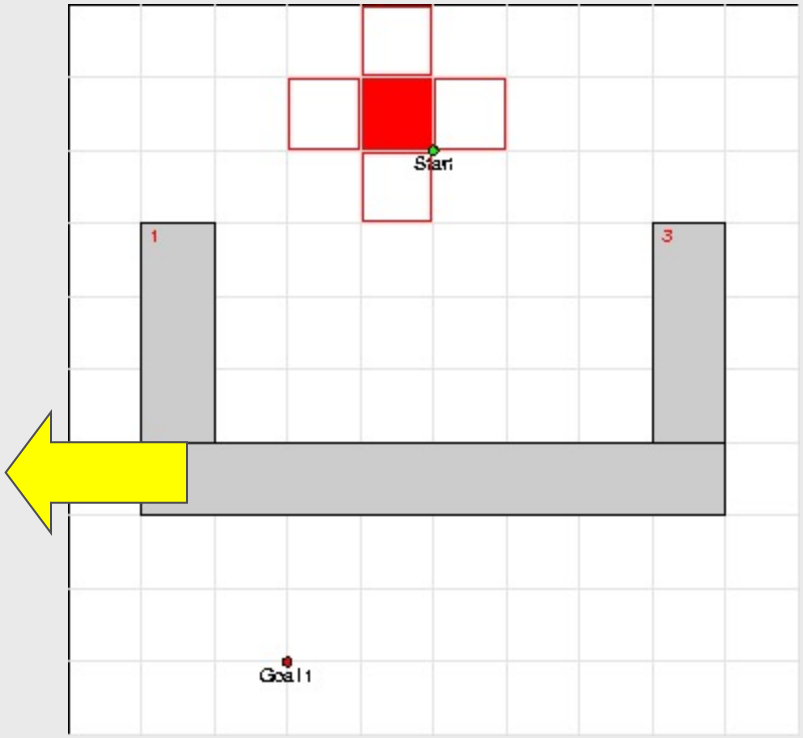


A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

$$f(n) = g(n) + h(n)$$

```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```

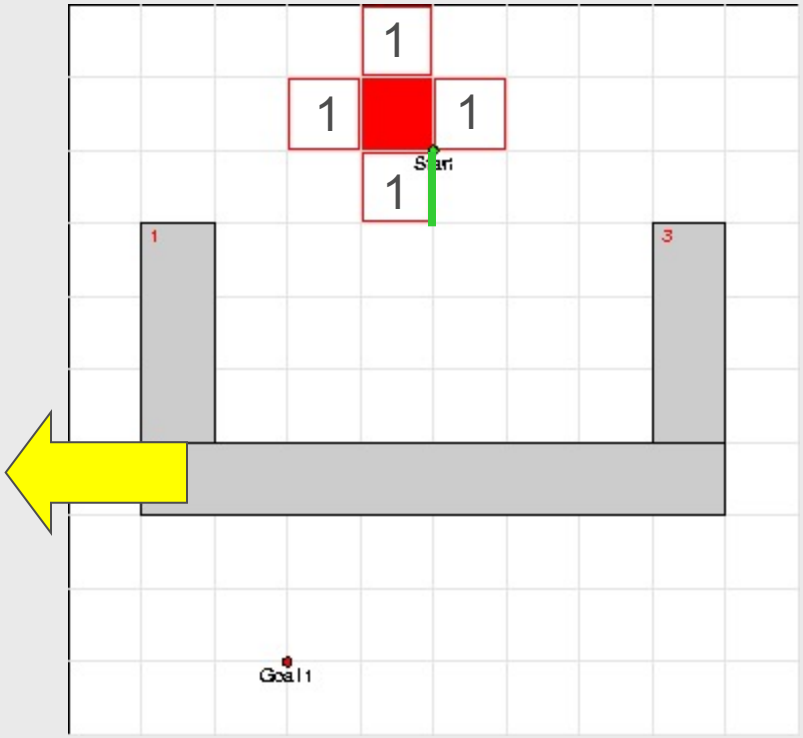


A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

  f(n) = g(n) + h(n)

```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```



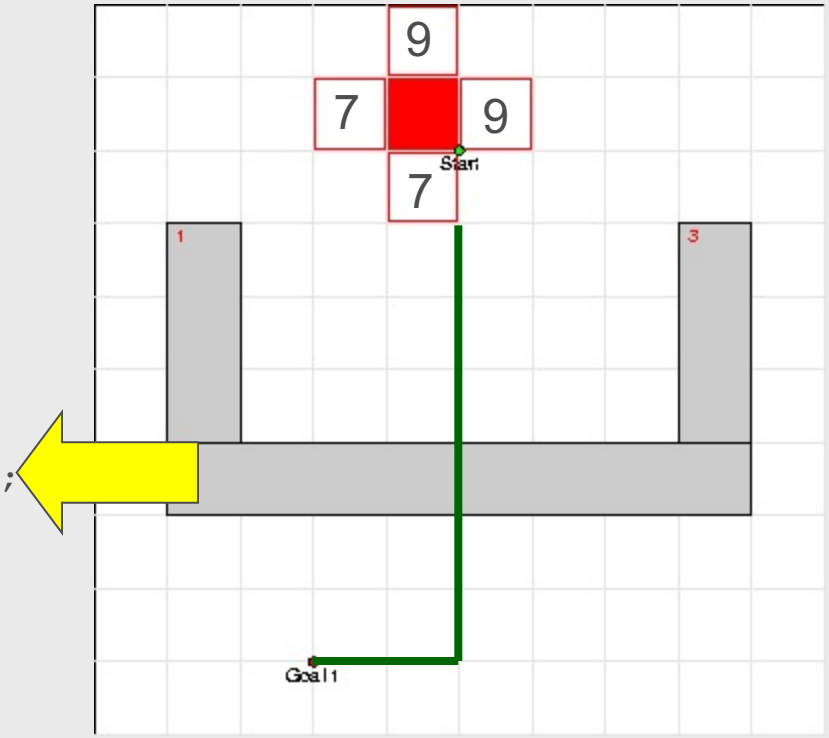
A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

  $f(n) = g(n) + h(n)$

  - Manhattan

```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```



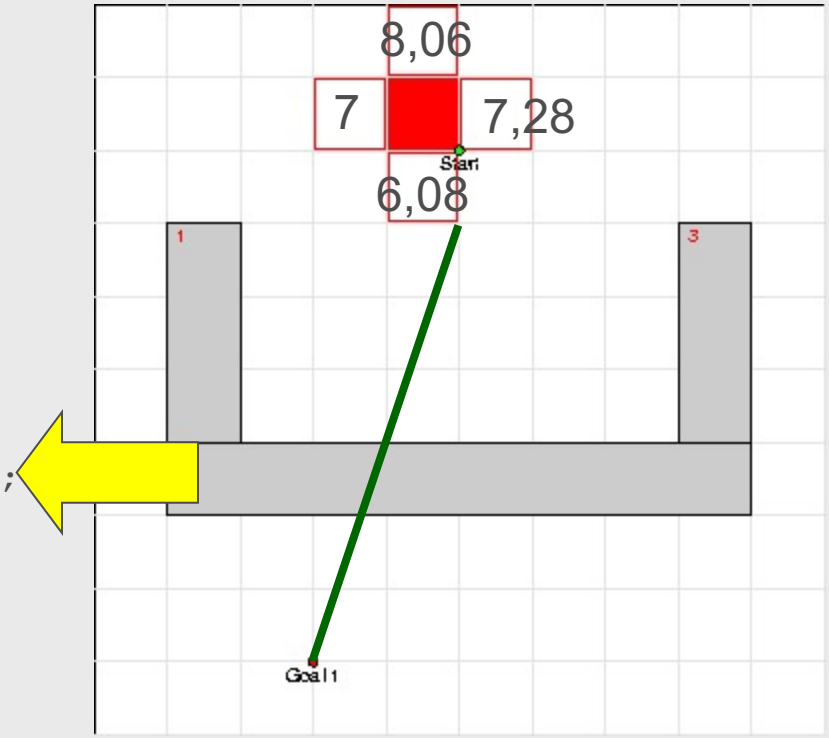
A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

  $f(n) = g(n) + h(n)$

  - Euclid

```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```

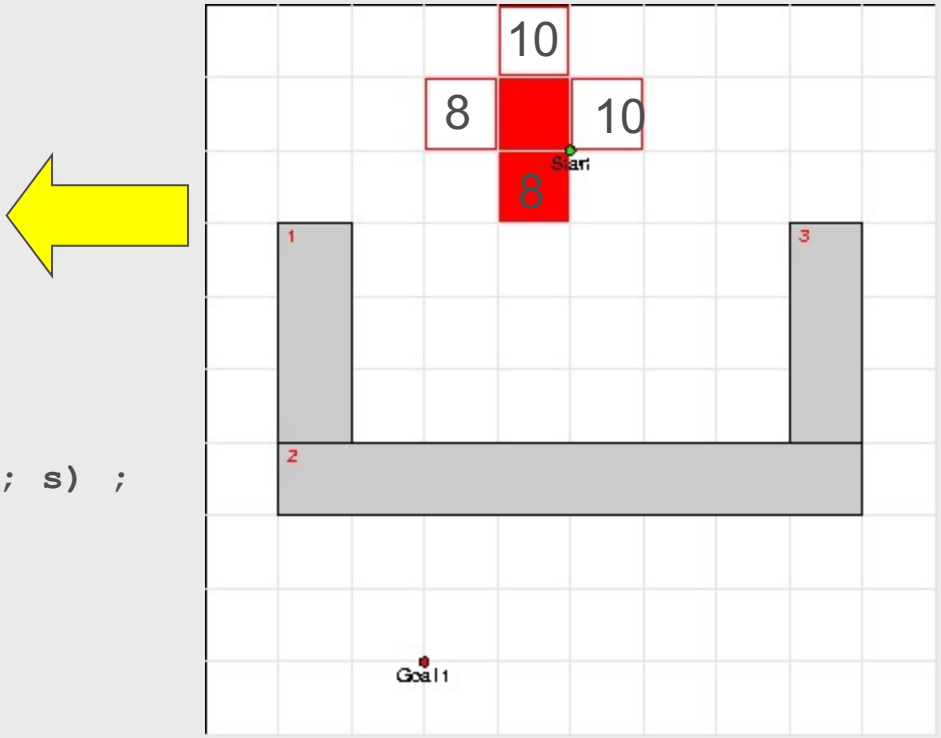


A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

$$f(n) = g(n) + h(n)$$

```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```

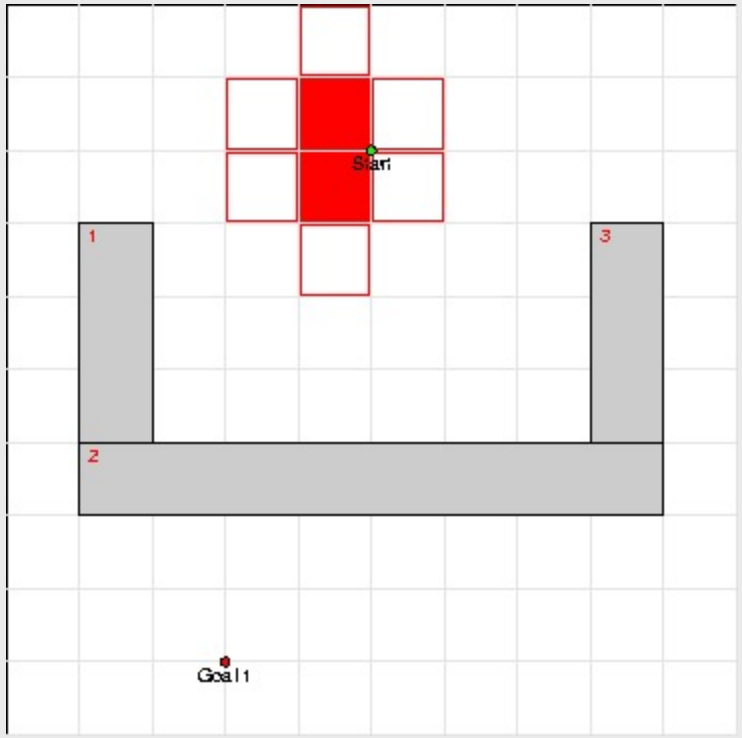


A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

$$f(n) = g(n) + h(n)$$

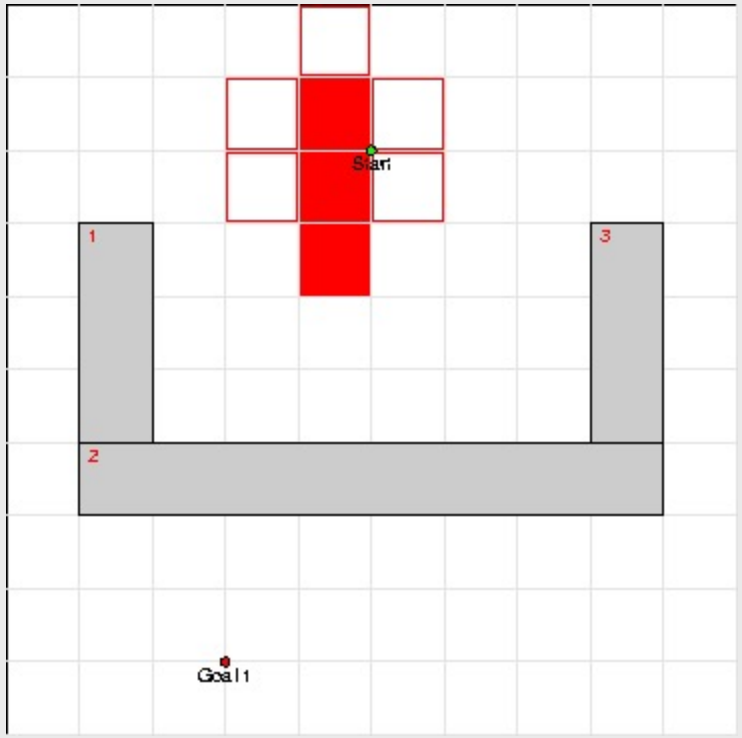
```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
```



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

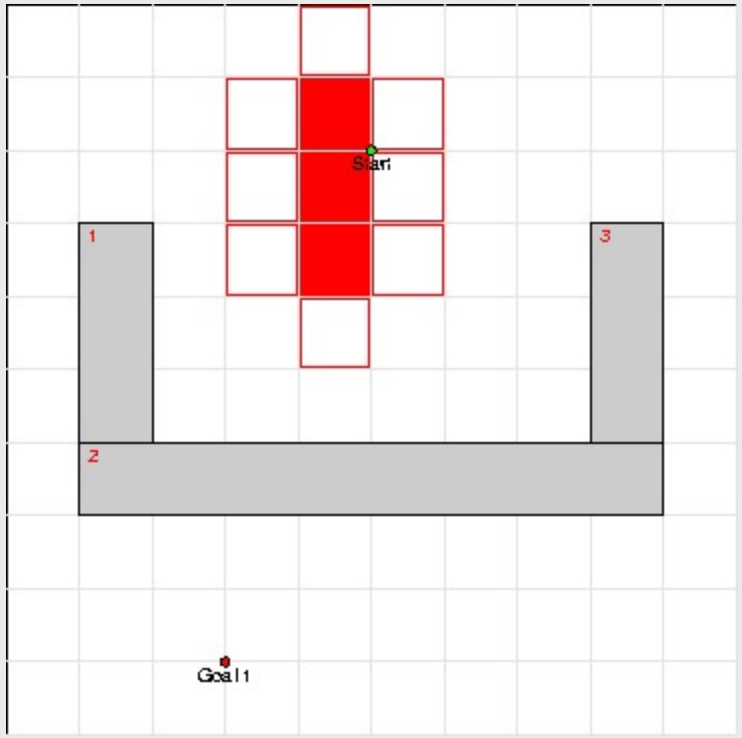
  $f(n) = g(n) + h(n)$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

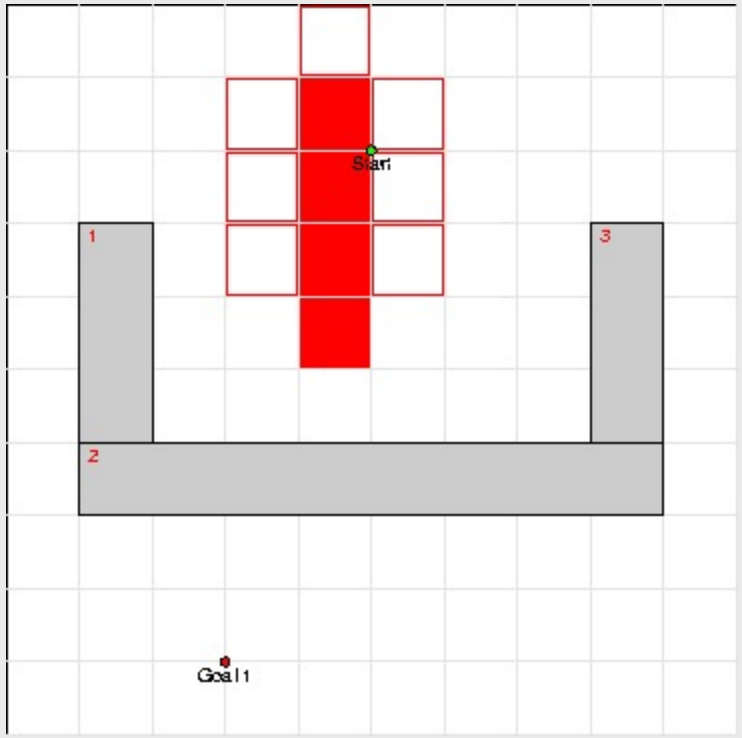
  $f(n) = g(n) + h(n)$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

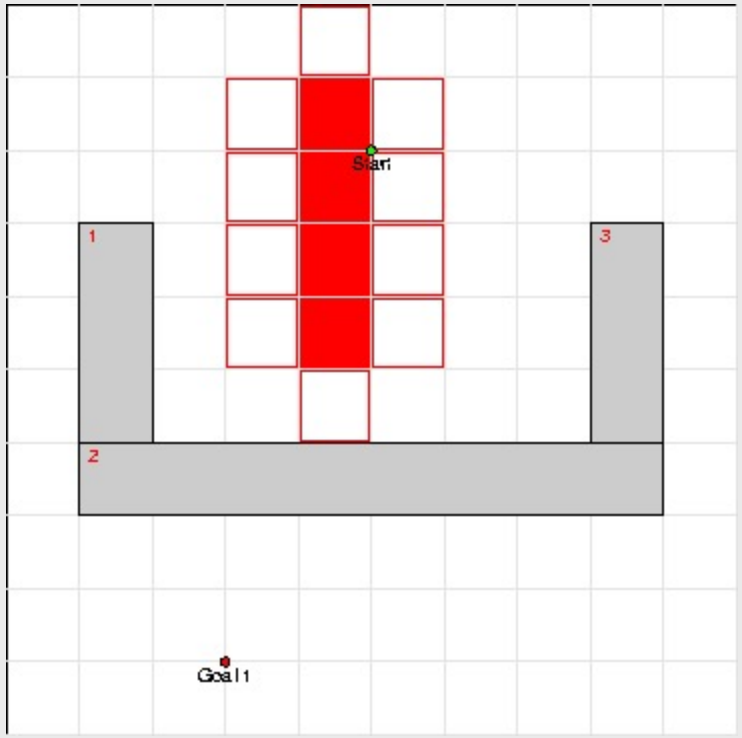
$$f(n) = g(n) + h(n)$$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

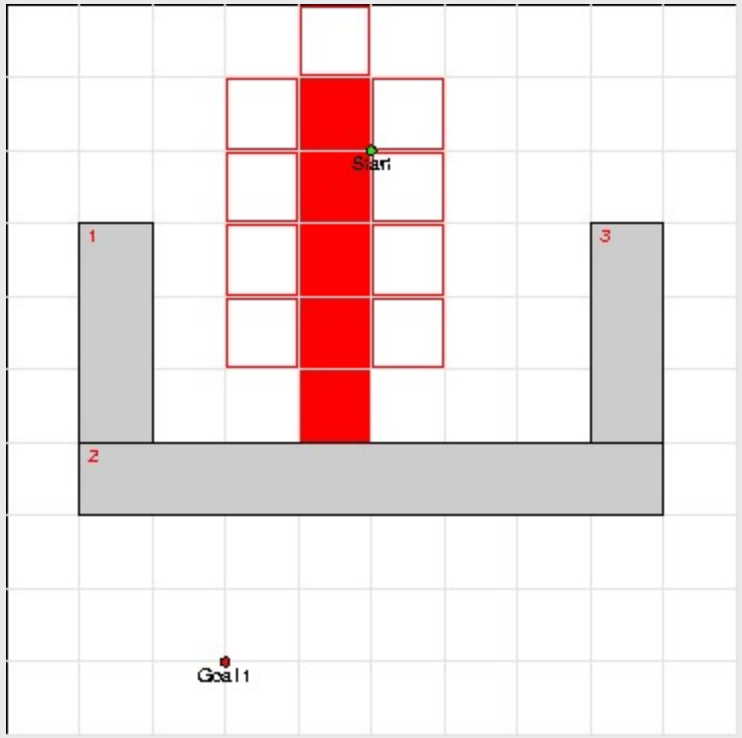
  $f(n) = g(n) + h(n)$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

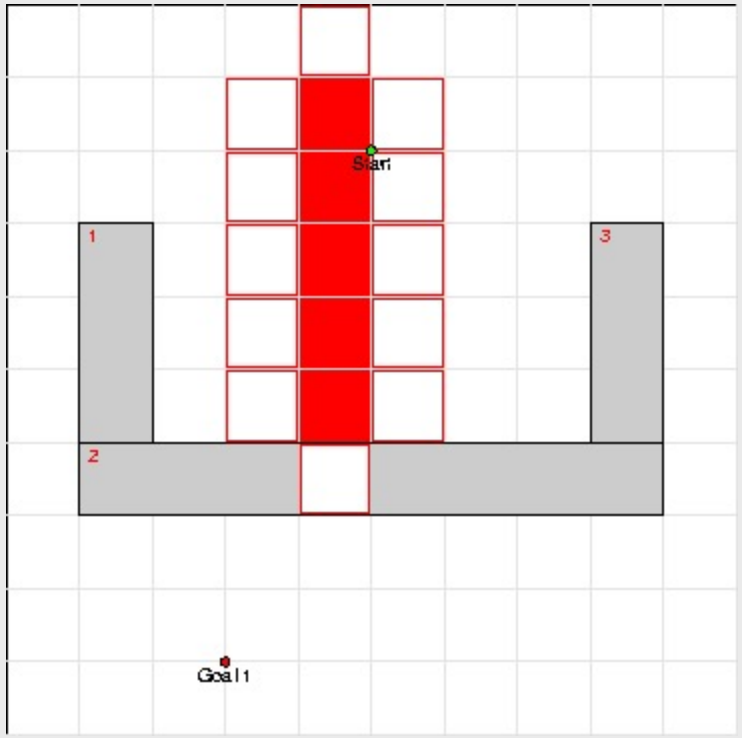
$$f(n) = g(n) + h(n)$$



A*-search

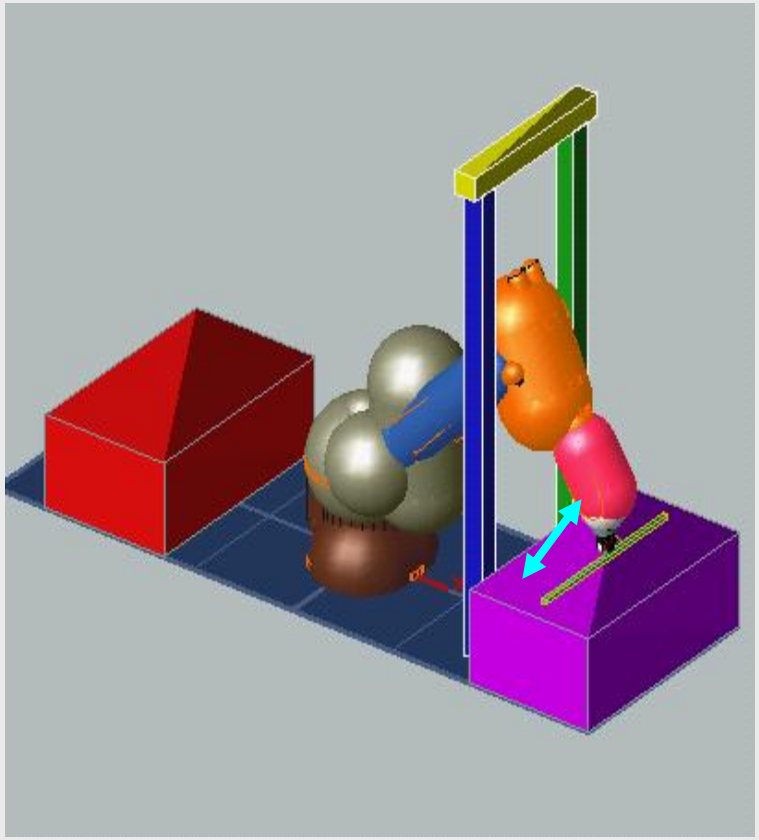
- Graph search (A*) in implicit C-space
- Heuristic evaluation function

  $f(n) = g(n) + h(n)$



Collision detection

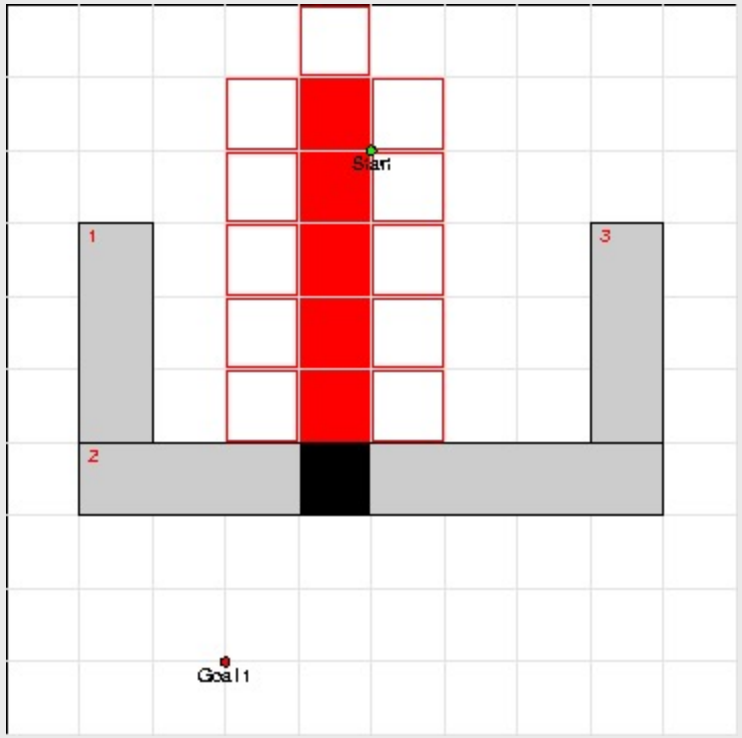
- Convex hull of robot and obstacles
- Approximation with primitive
- Distance-Calculation with Gilbert-Johnson-Kerthi-Algorithm



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

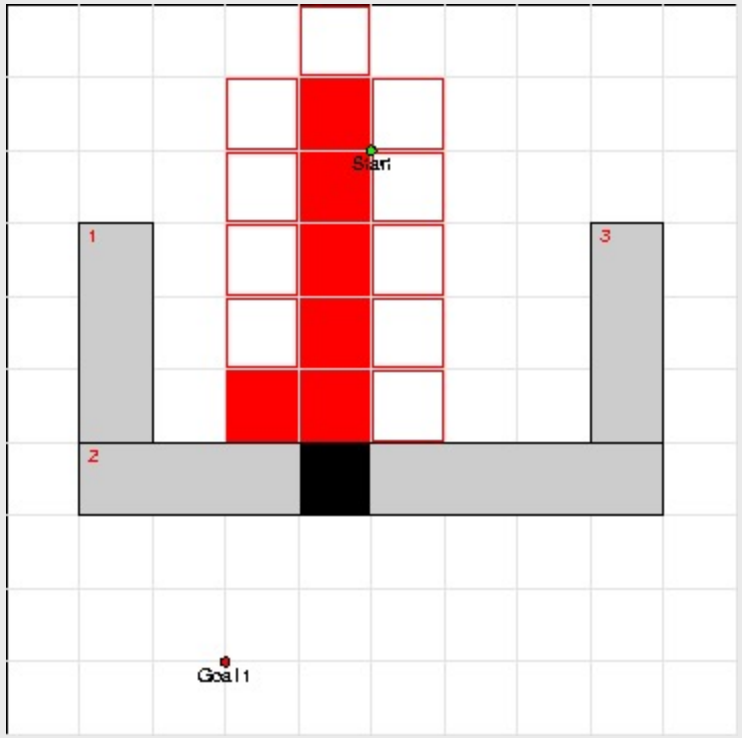
$$f(n) = g(n) + h(n)$$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

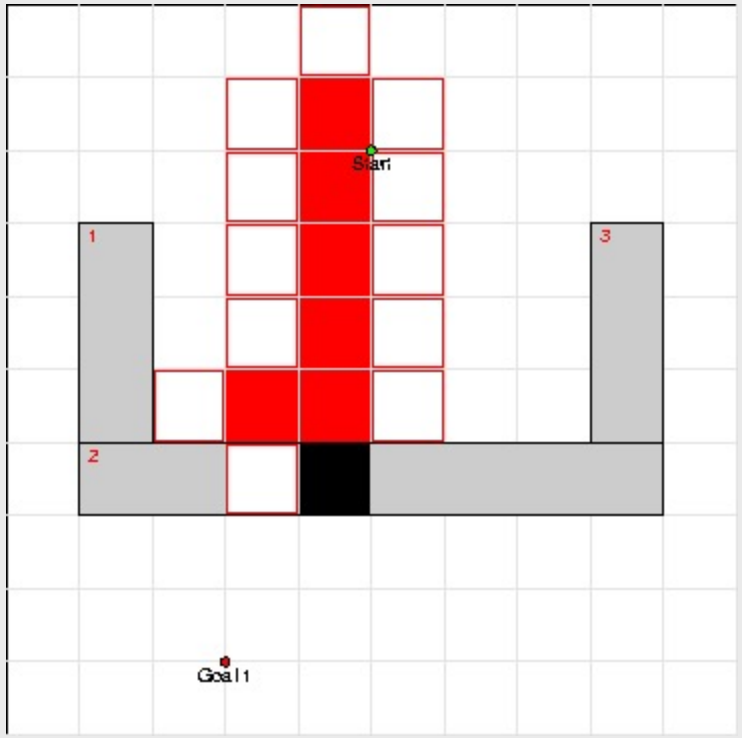
  $f(n) = g(n) + h(n)$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

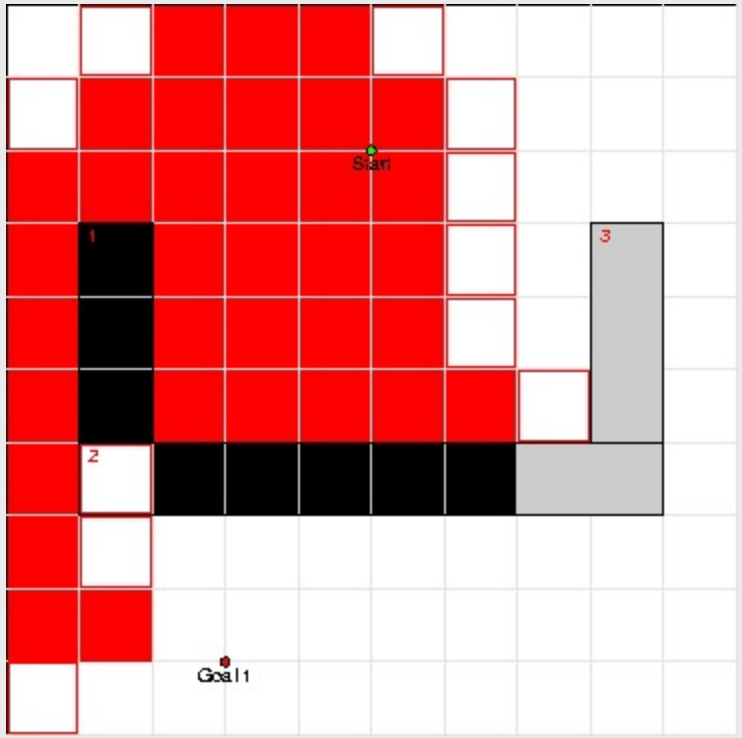
  $f(n) = g(n) + h(n)$



A*-search

- Graph search (A*) in implicit C-space
- Heuristic evaluation function

  $f(n) = g(n) + h(n)$

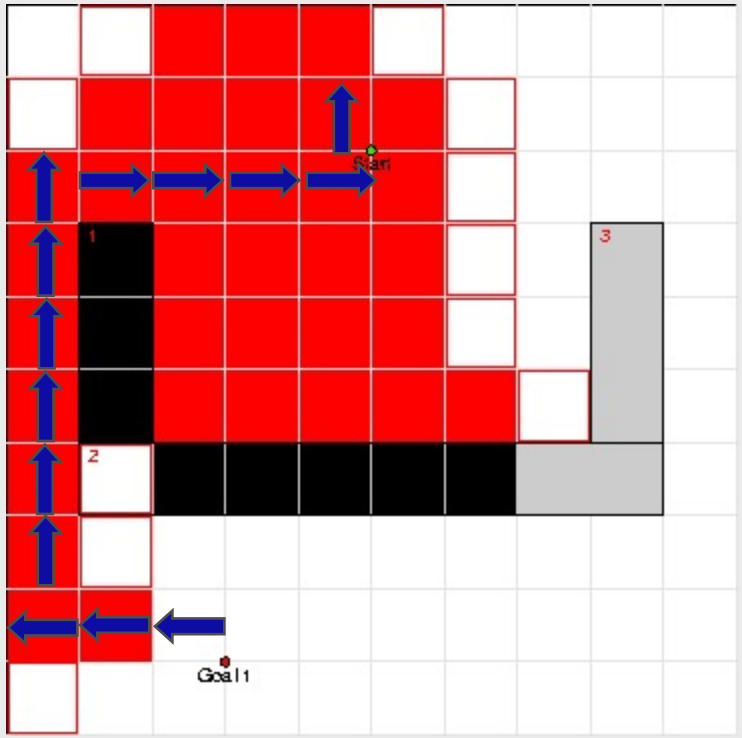


A*-search

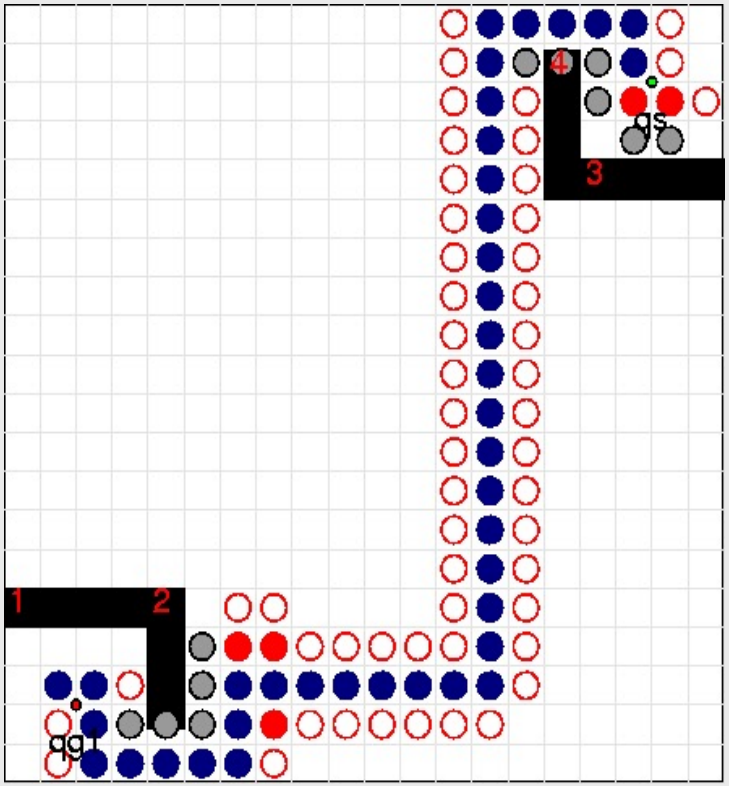
- Graph search (A*) in implicit C-space
- Heuristic evaluation function

  $f(n) = g(n) + h(n)$
- Collecting solution

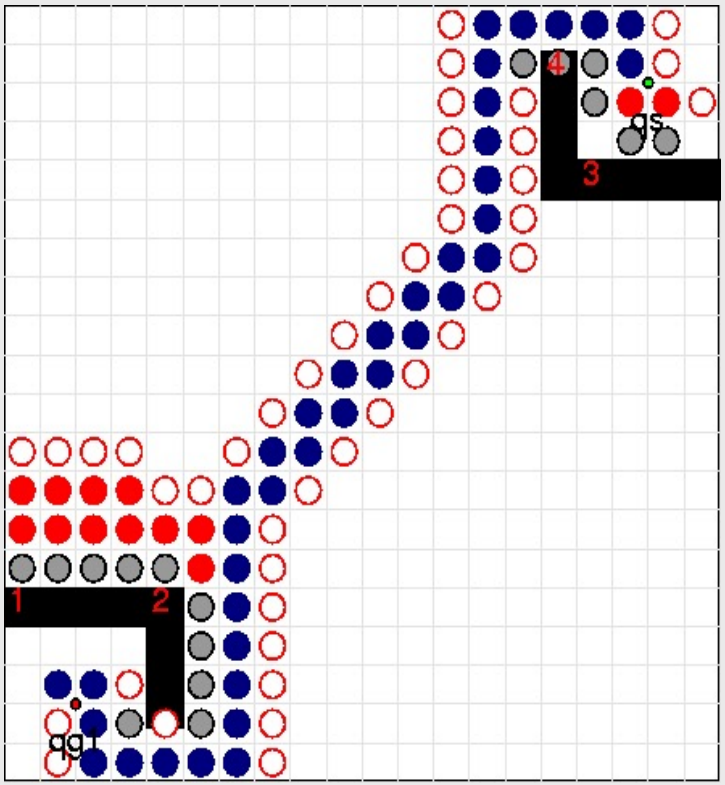
```
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ ∅) do begin
    n = best (OPEN) ;
    if (n == G) then
         goalFound = TRUE;
         break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
if (goalFound == TRUE) then
         printSolution (: : : ) ;
```



Comparison: Used Metrics



MANHATTAN



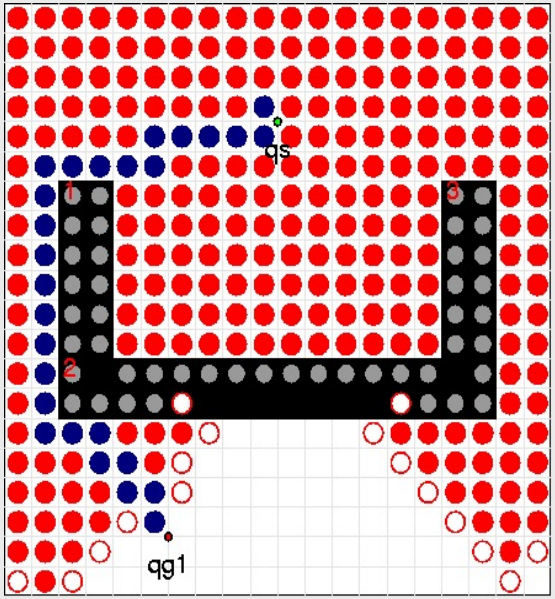
EUKLID

Modifying evaluation function

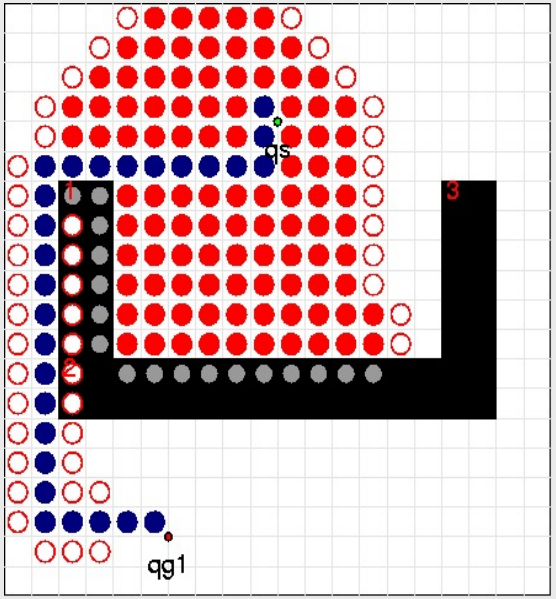
- Modified evaluation function:

  f(n) = (1-w) * g(n) + w * h(n)

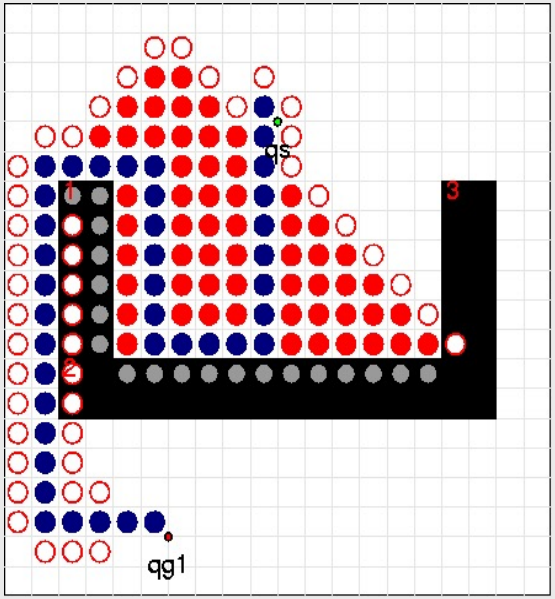
  - w : Heuristic weight



w = 0

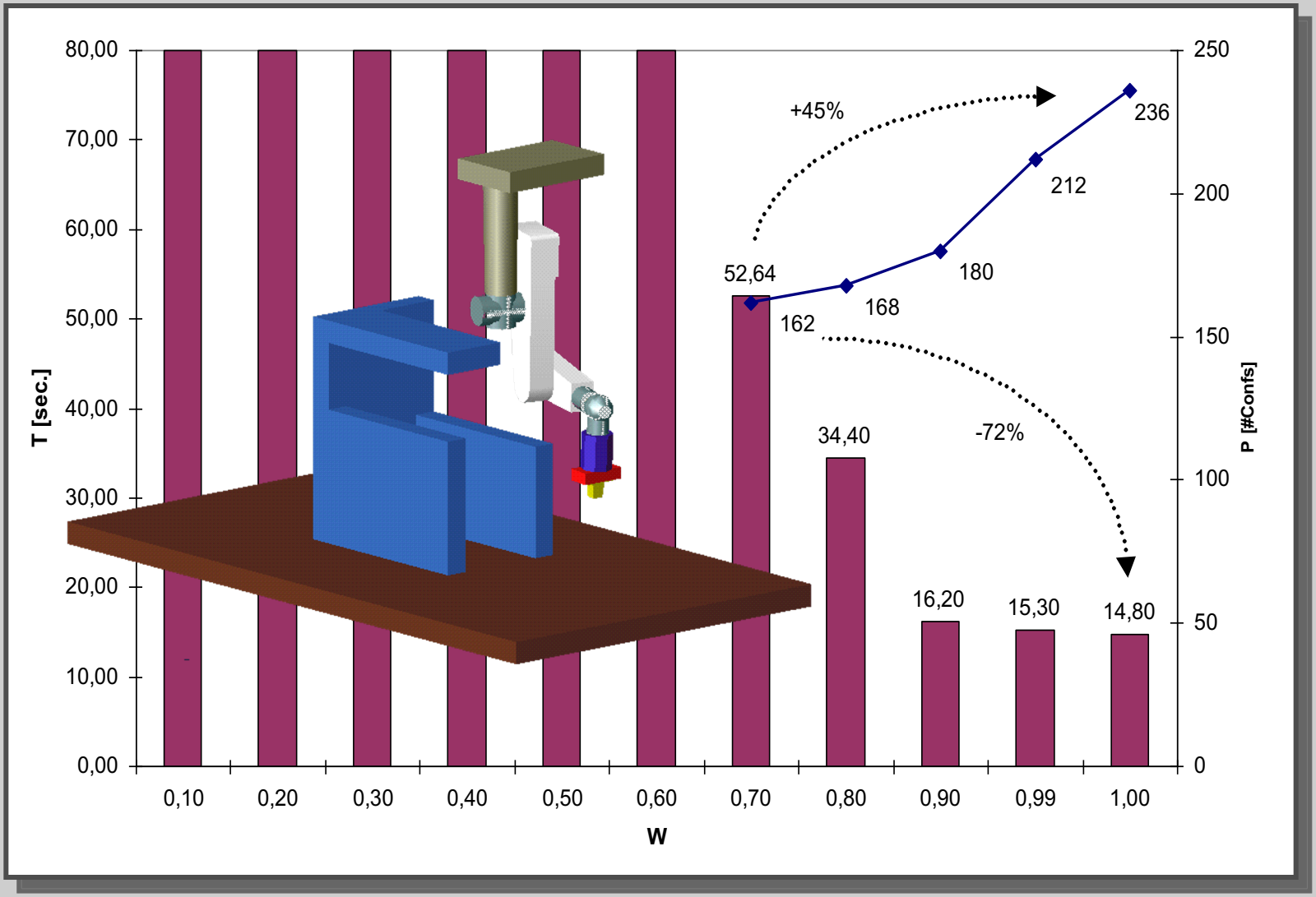


w = 0.5

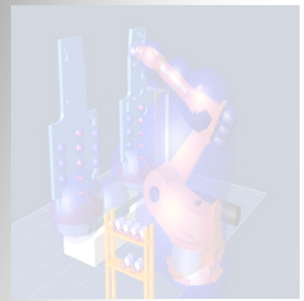


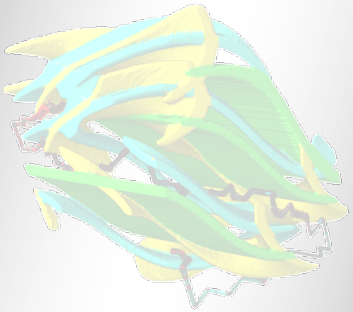
w = 1

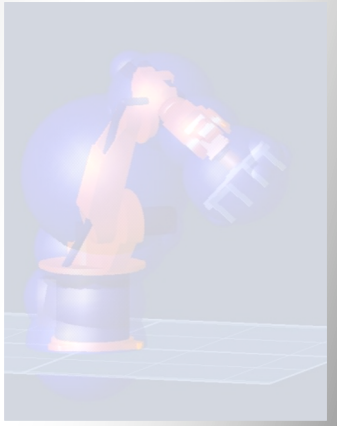
Heuristic weight (TRAP)



Extensions of A*

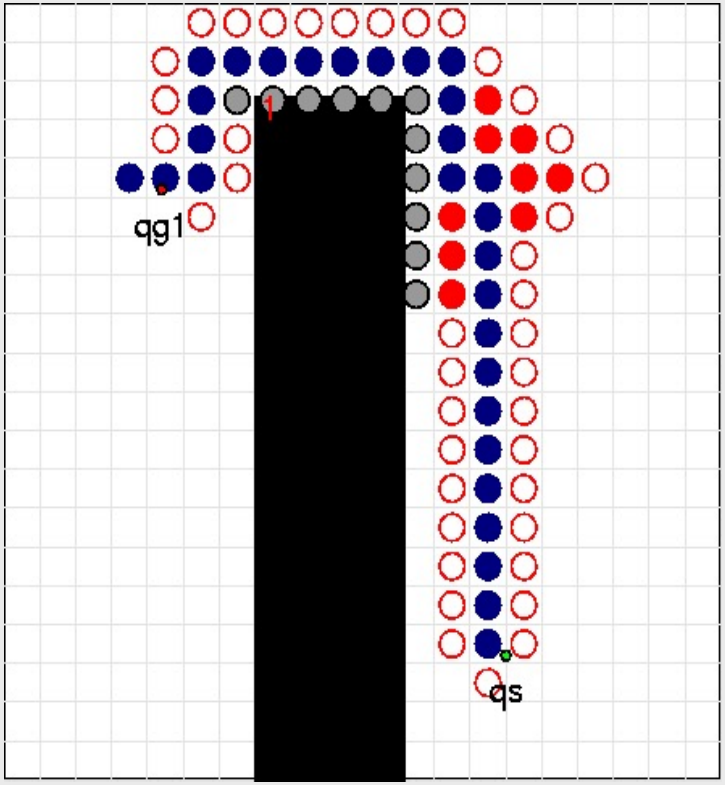






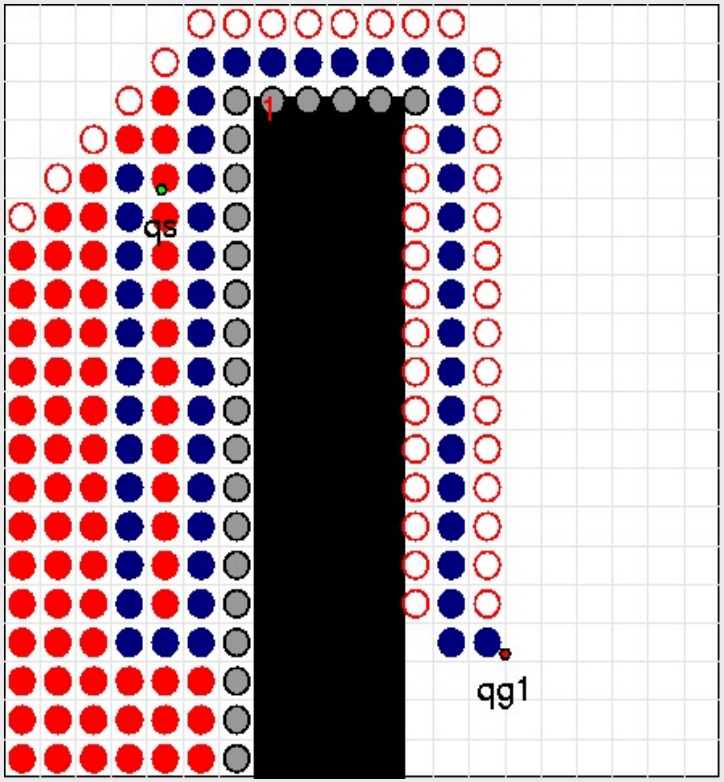
Unidirectional search

- Problem
  - Time needed to find solution is depending on **search direction**



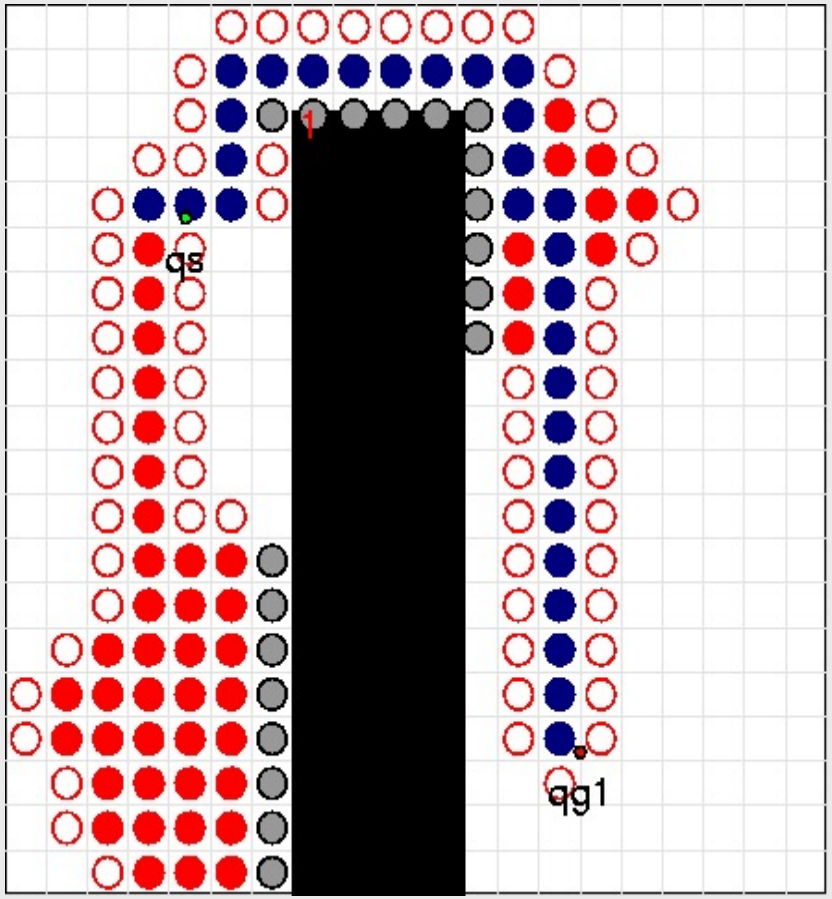
Other direction

- Problem
  - Time needed to find solution is depending on **search direction**



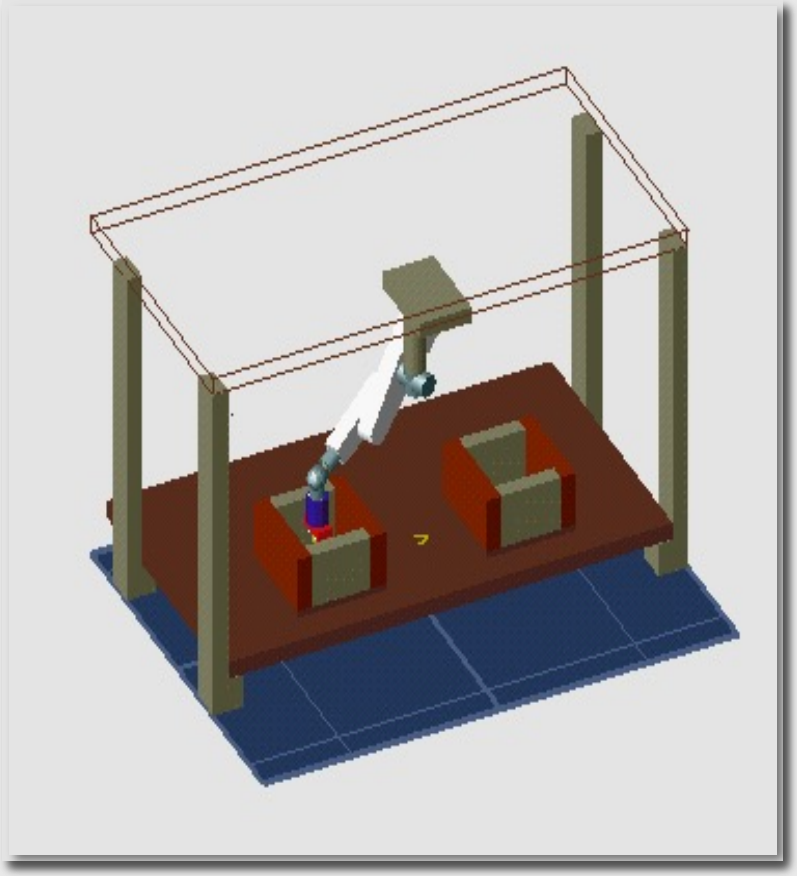
Bidirectional search

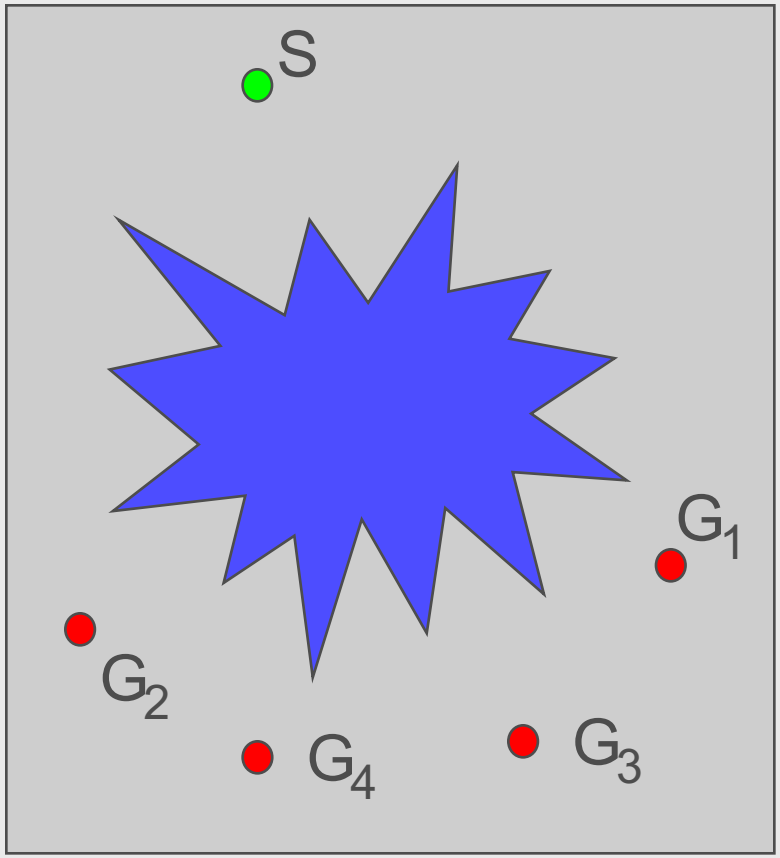
- **Goal:** combining forward and backward search
- Modifying A*-algorithm:
  - **Two OPEN-Lists** covering forward- and backward search
  - **Nodes of the two lists** are alternatingly expanded
  - Search algorithm ends, if a duplicate in shared CLOSED-List is found



Inverse kinematics: „More than one goal“

- Benchmark „STAR“:
  - Goal configuration can be reached in 4 different ways





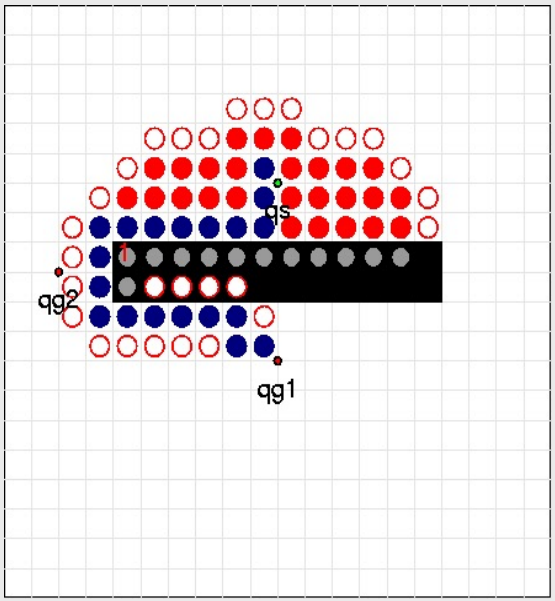
- How can this be used during search?

Dynamic goal switching

Idea

- include all possible goal configurations given by inverse kinematics during search
- In `handleSuccessors(...)` heuristic value to all goals are evaluated
  - Search direction is always heading for the nearest goal
    - Current goal will automatically change during search if it seems better suited

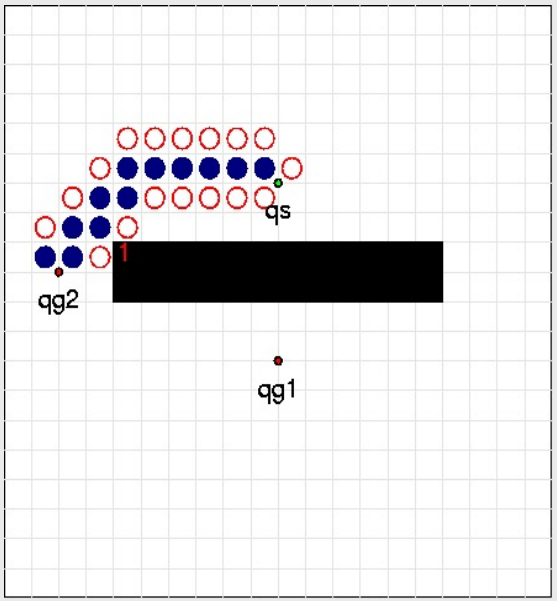
Dynamic goal switching



$q_s \rightarrow q_{g1}$,

no goal switching

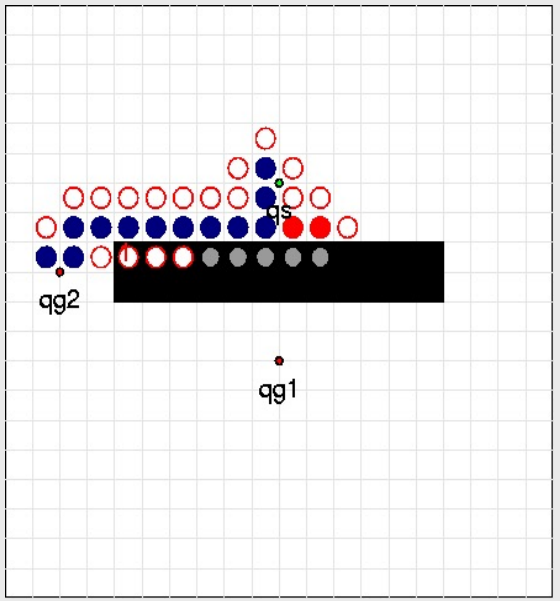
Nodes: 55



$q_s \rightarrow q_{g2}$,

no goal switching

Nodes: 12



$q_s \rightarrow q_{g1}$ & q_{g2},

goal switching

Nodes: 19

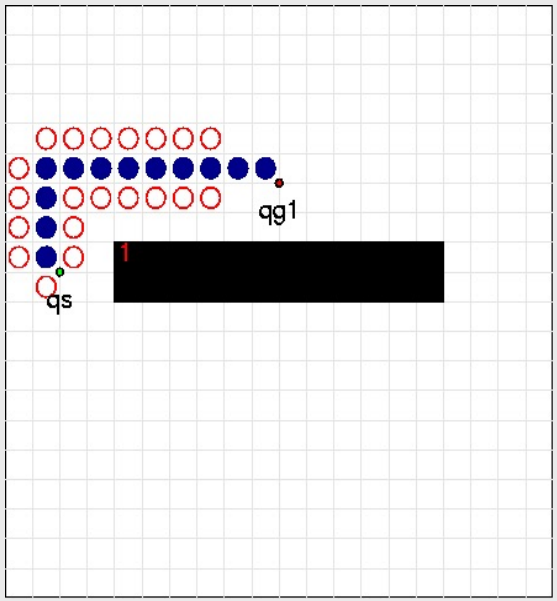
Dynamic start switching

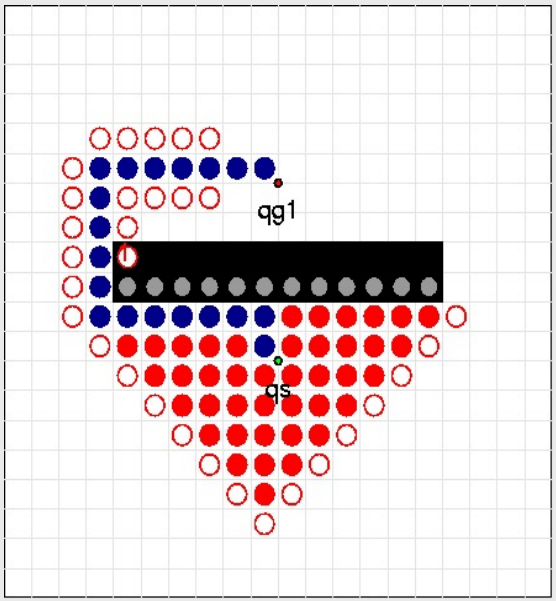
- Goal switching works fine!
- Use same thing for start

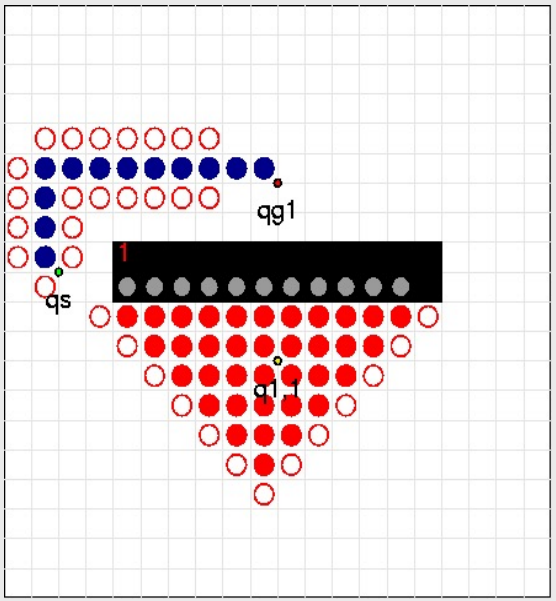
Idea

- Insert all start configurations into OPEN-List
- Let A* automatically choose best one

Dynamic start switching







$q_s \rightarrow q_{g1}$,

no start switching

Nodes: 12

$q_{s2} \rightarrow q_{g2}$,

no start switching

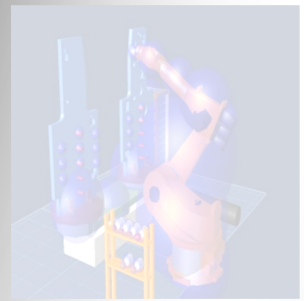
Nodes: 72

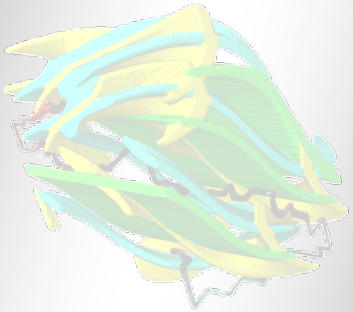
q_{s1} & $q_{s2} \rightarrow q_{g1}$,

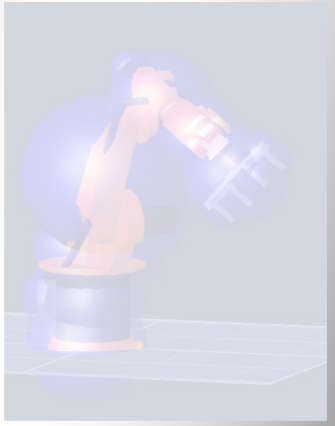
start switching

Nodes: 59

Parallel A*-Search

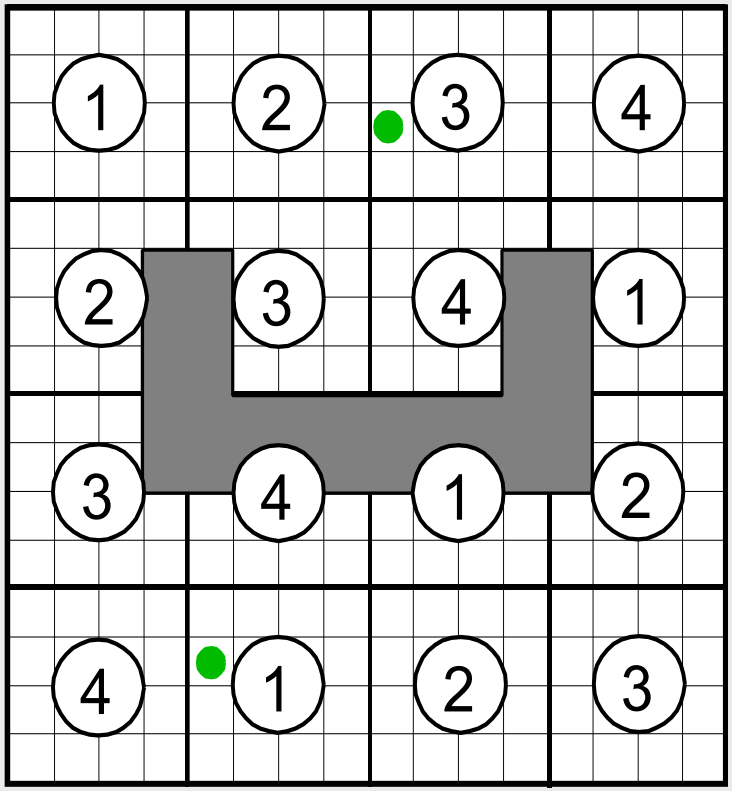


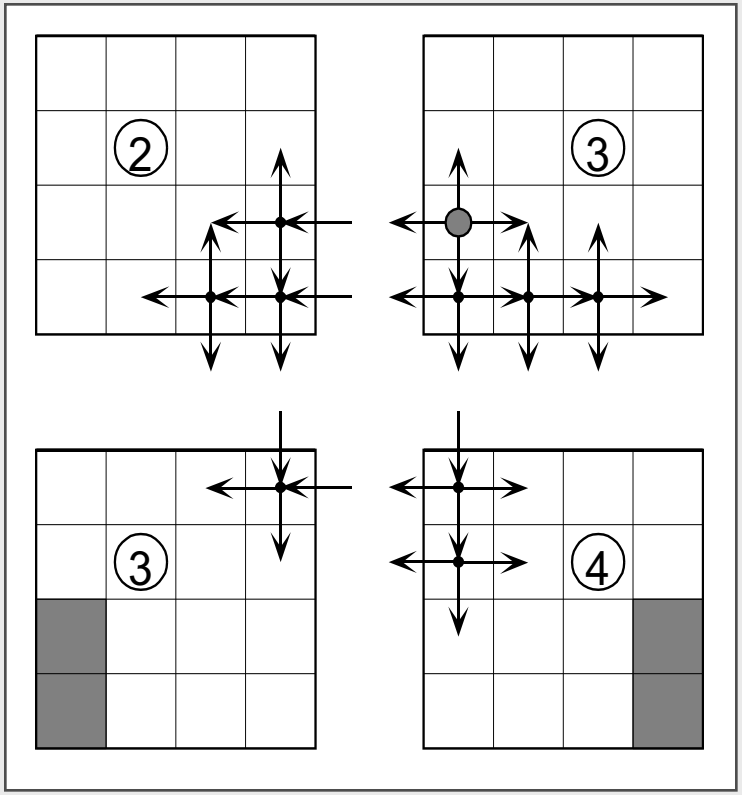




Parallel A*-Search: Idea

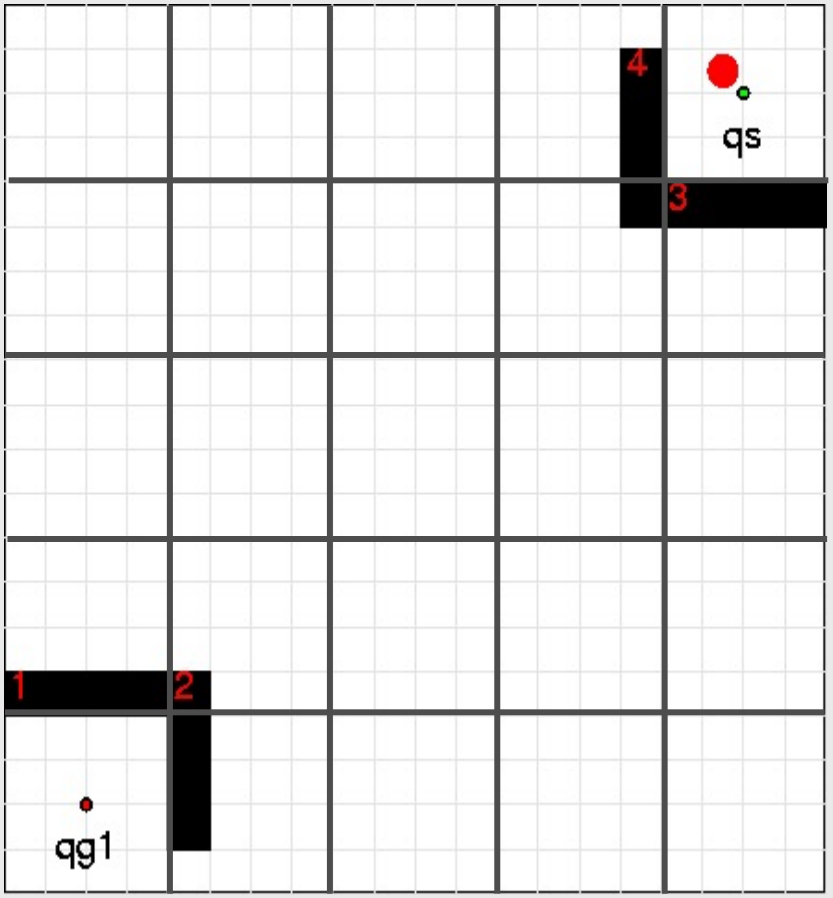
- C-space can easy separated
- Search is done on more than one processor
- Speed up planning process





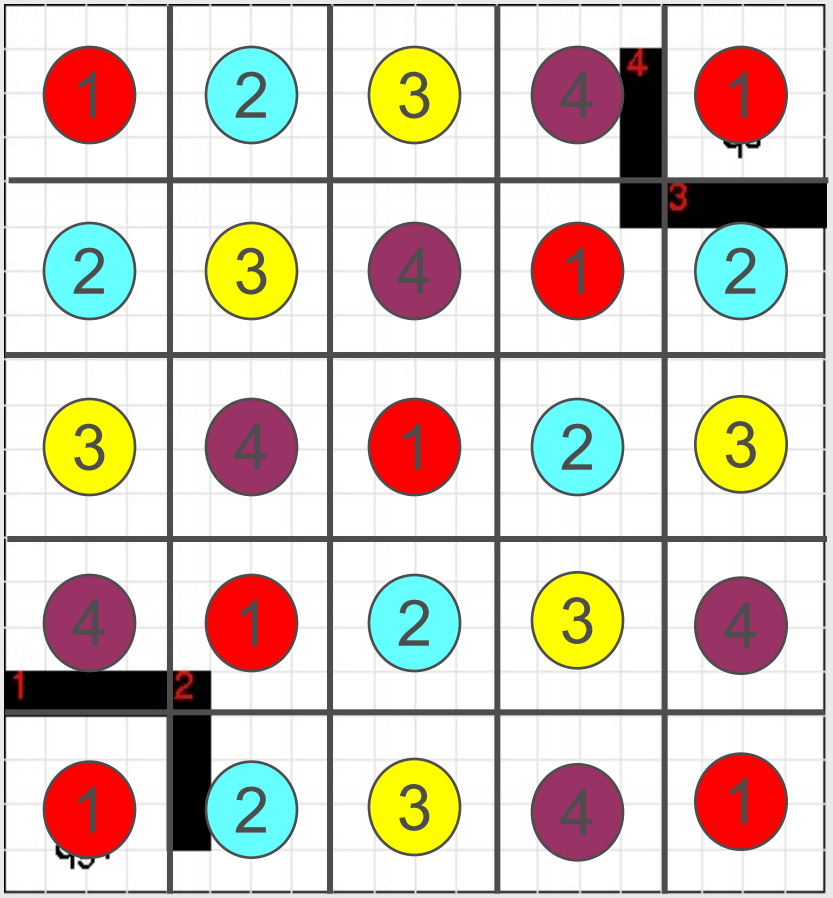
Parallel A*-Search

- Separation in hypercubes



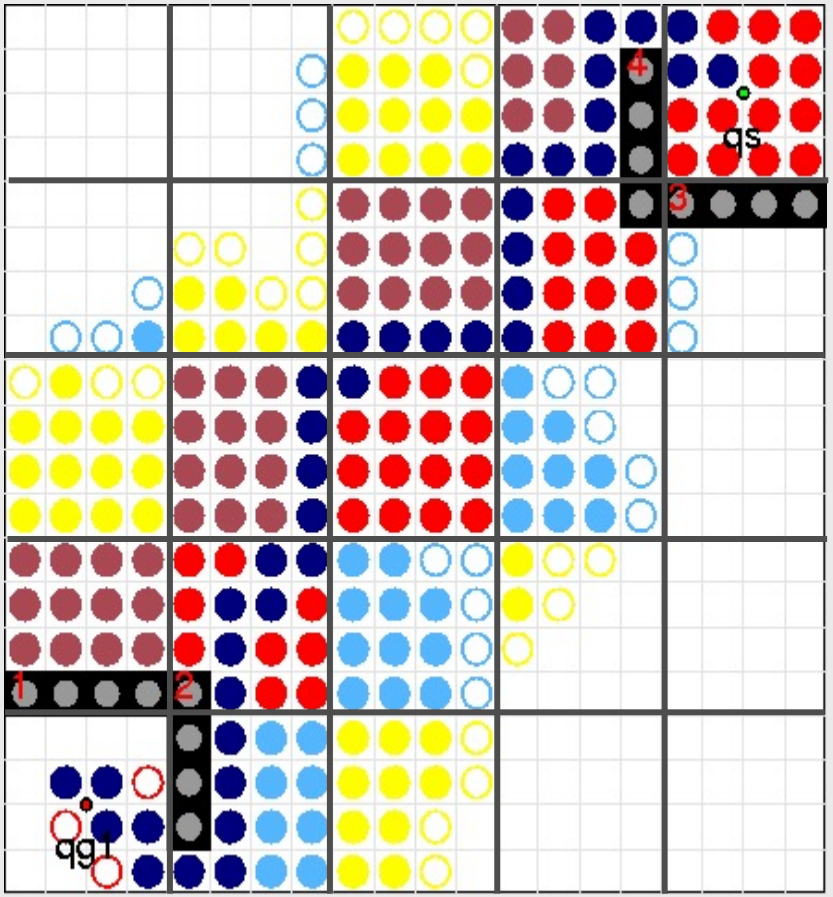
Parallel A*-Search

- Separation in hypercubes
- cyclic distribution of processors



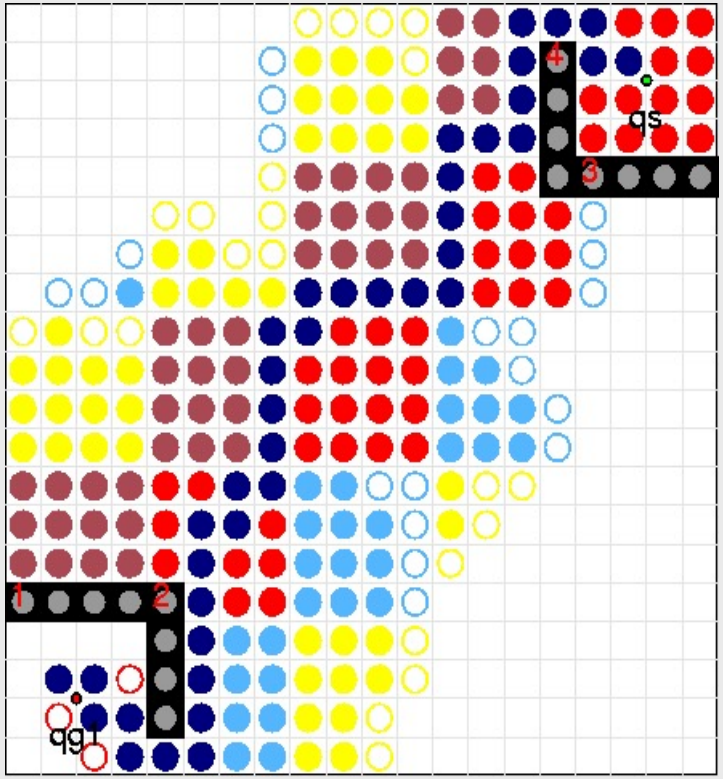
Parallel A*-Search

- Separation in hypercubes
- cyclic distribution of processors
- Local A*-Search on every processor

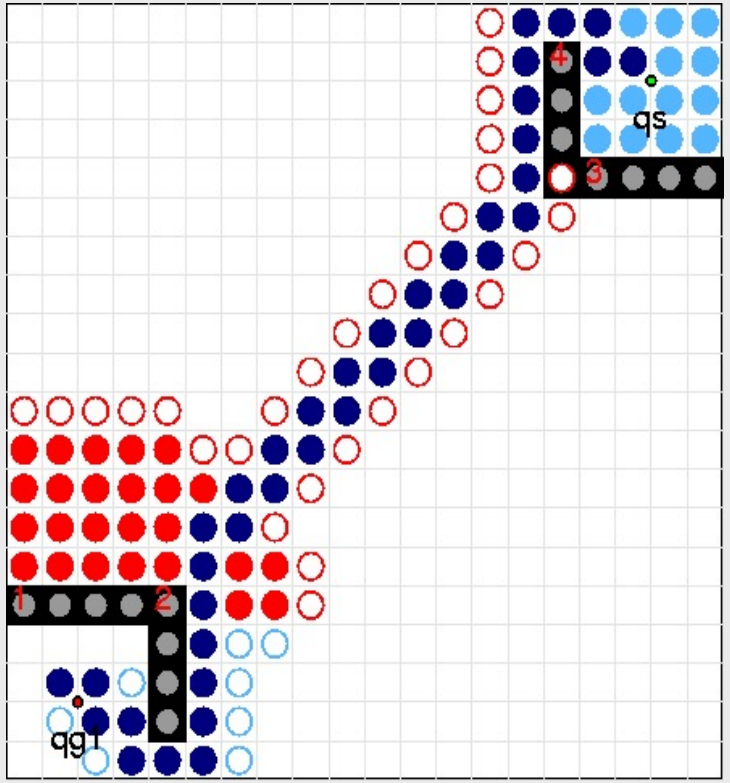


Parallel A*-Search: Size of hypercubes

- Solution is depending on size of hypercubes

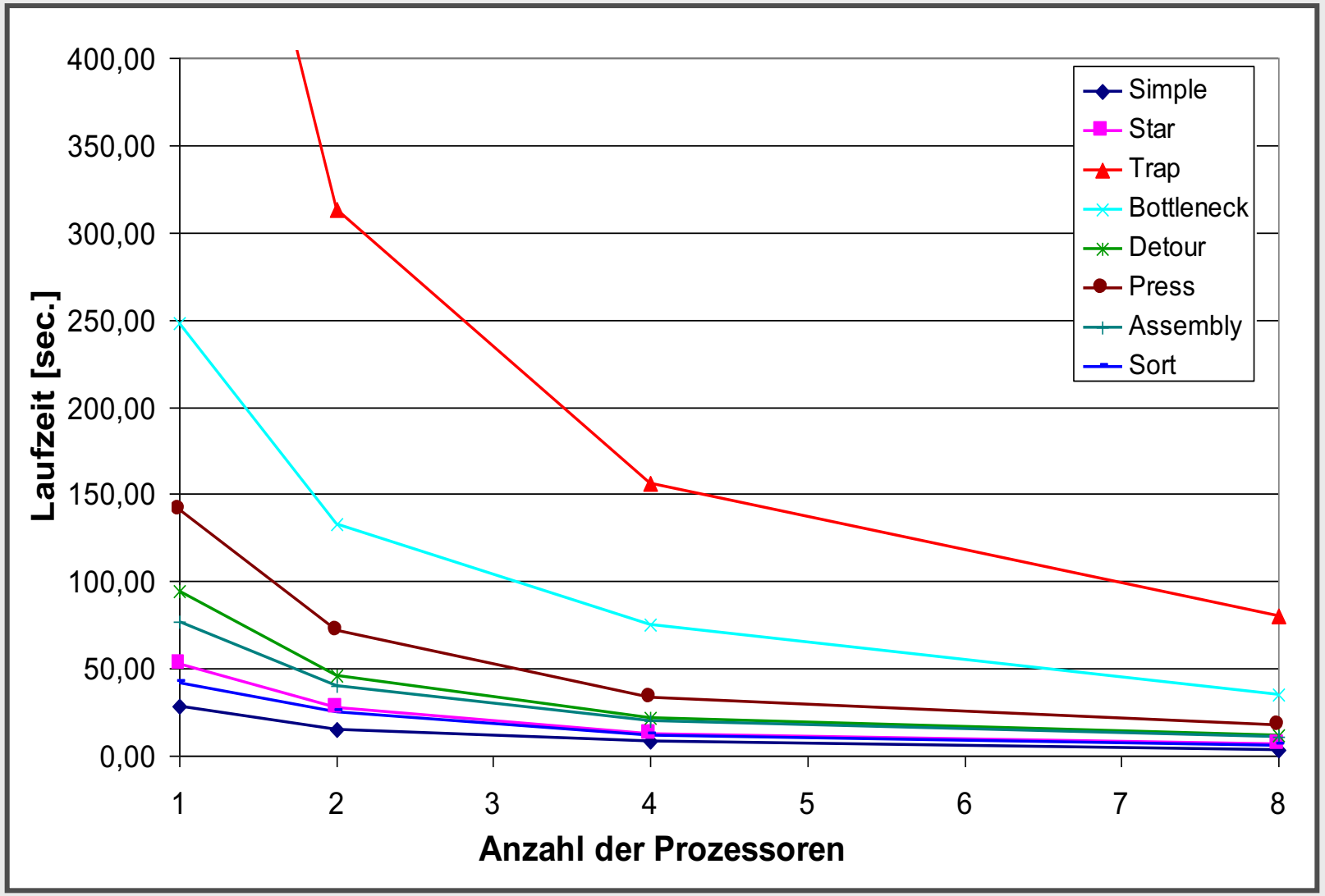


b=4



b=16

Planning time with parallel A*-Search



Speedup

- Speedup $S = T_{serial}/T_{parallel}$