Designing Push Plans for Disk-Shaped Robots

Dirk H.P. Gerrits
dirk@dirkgerrits.com

28 May 2008

/ department of mathematics and computer science

TU/e Technische Universiteit Eindhoven University of Technology

Contents

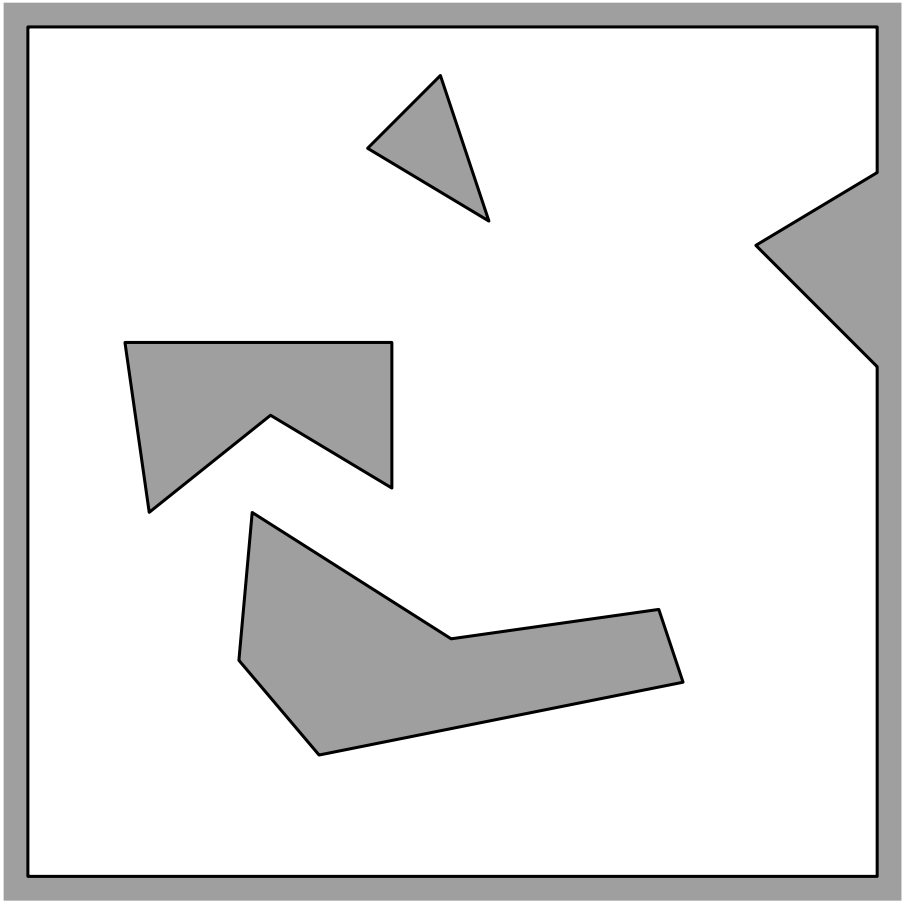
- Context & problem description
- Our new algorithm
- New contributions & future research

Path planning

A fundamental problem in robotics.

Path planning

A fundamental problem in robotics.

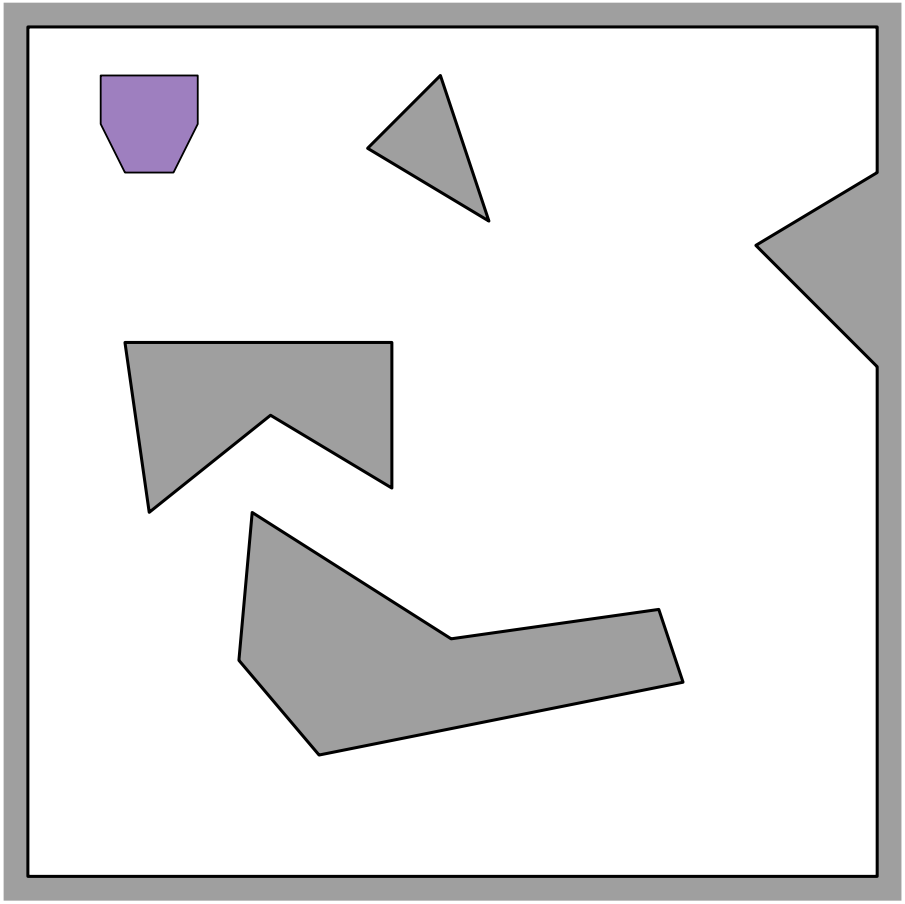


Given:

- A collection of obstacles.

Path planning

A fundamental problem in robotics.

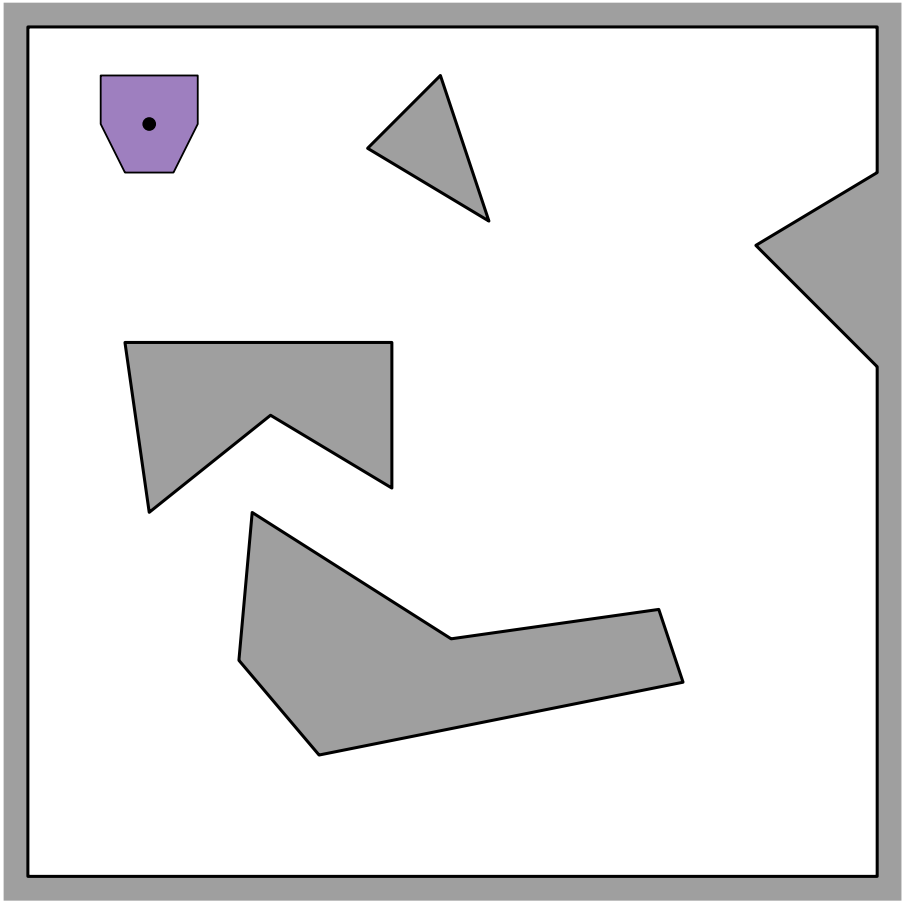


Given:

- A collection of obstacles.
- A robot

Path planning

A fundamental problem in robotics.

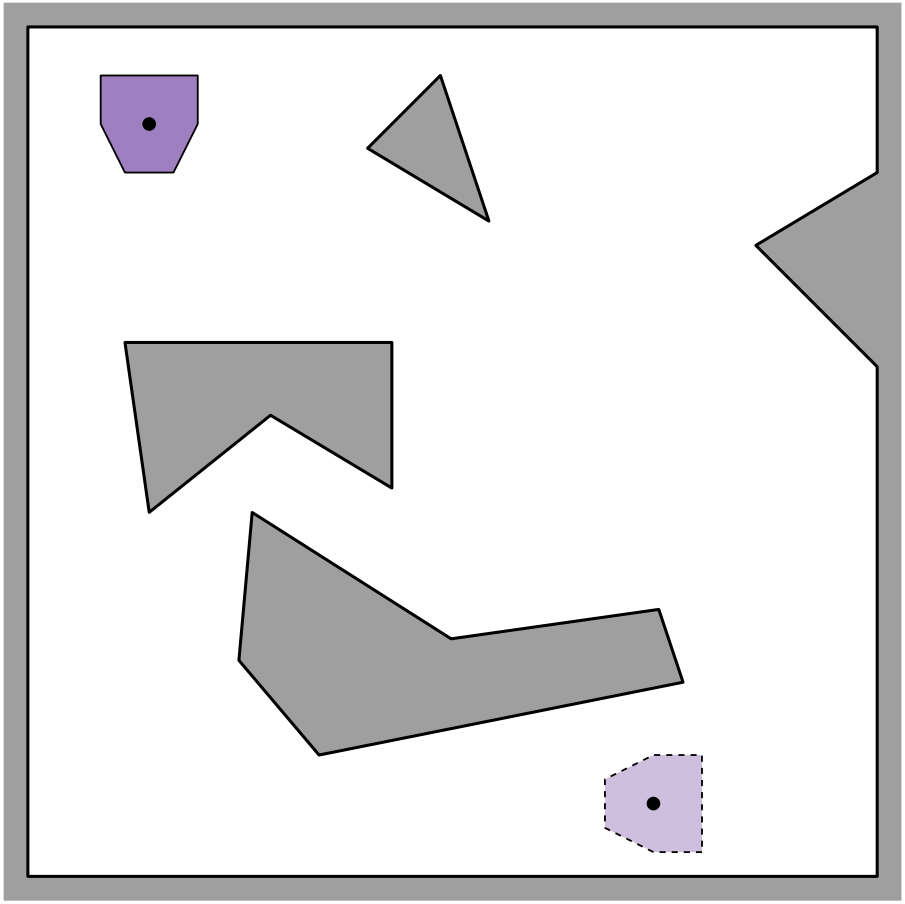


Given:

- A collection of obstacles.
- A robot at initial configuration.

Path planning

A fundamental problem in robotics.

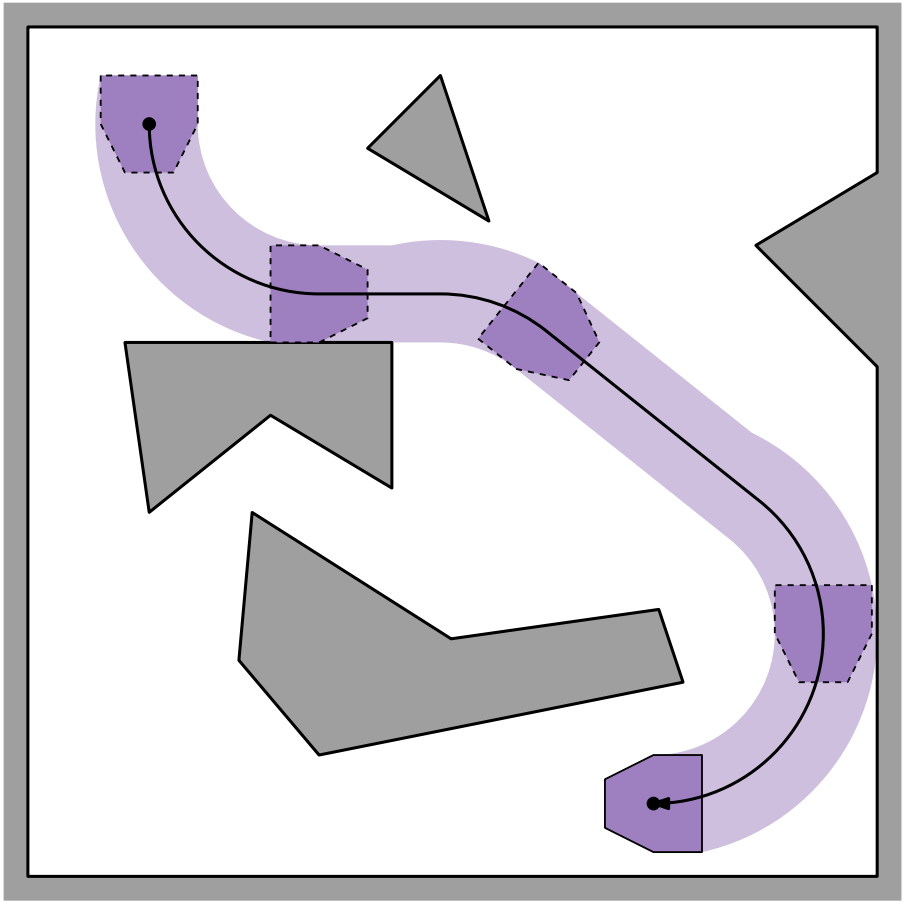


Given:

- A collection of obstacles.
- A robot at initial configuration.
- A destination configuration.

Path planning

A fundamental problem in robotics.



Given:

- A collection of obstacles.
- A robot at initial configuration.
- A destination configuration.

Find:

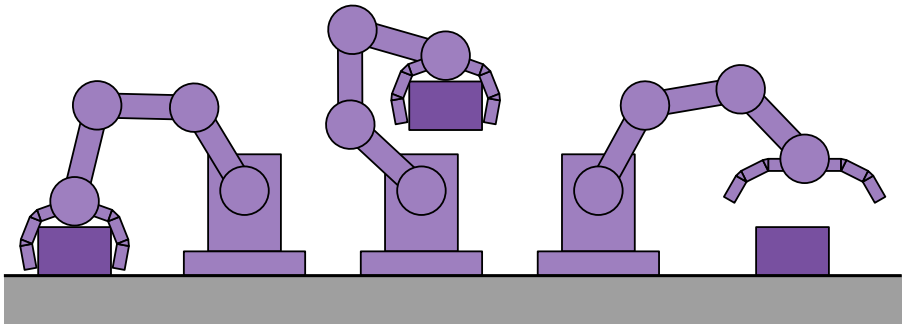
- A collision-free path from the initial configuration to the destination configuration. (Or report that none exists.)

Manipulation path planning

Now it's not the (active) robot that needs to reach a destination, but a (passive) object.

Manipulation path planning

Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:

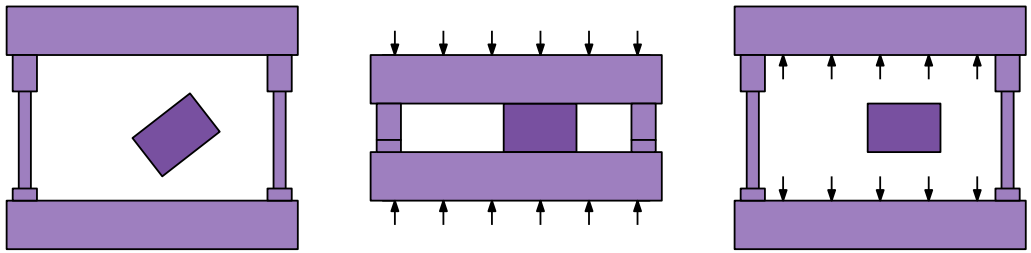


- Grasping

Manipulation path planning

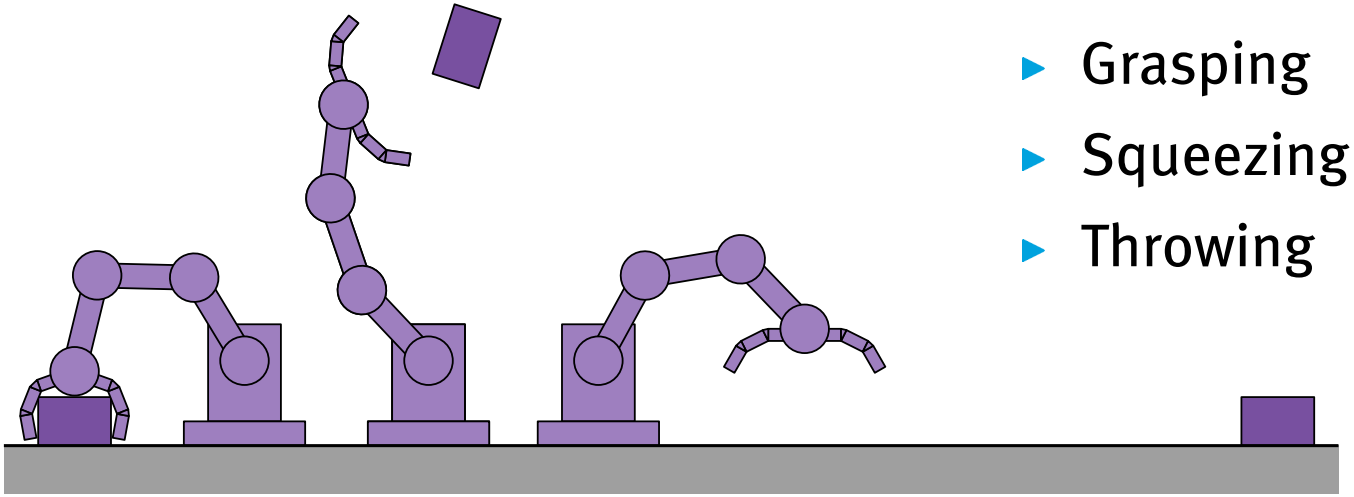
Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:

- Grasping
- Squeezing



Manipulation path planning

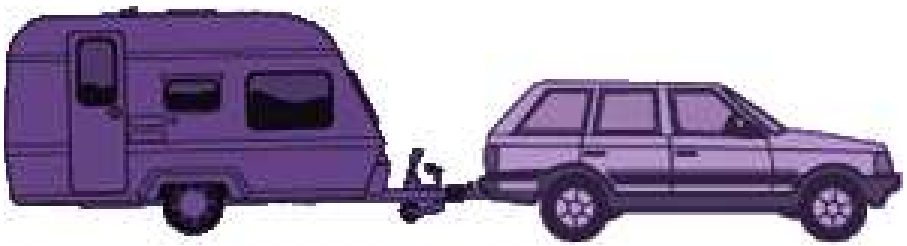
Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:



- Grasping
- Squeezing
- Throwing

Manipulation path planning

Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:



- Grasping
- Squeezing
- Throwing
- Pulling

Manipulation path planning

Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:

- Grasping
- Squeezing
- Throwing
- Pulling
- Pushing

Manipulation path planning

Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:

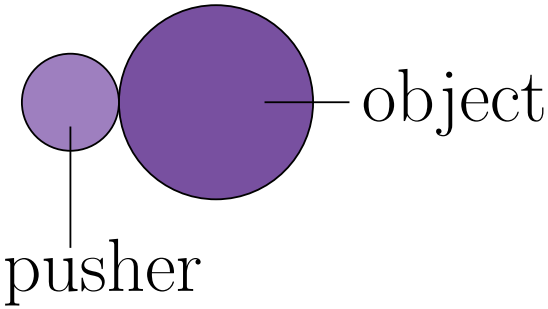
- Grasping
- Squeezing
- Throwing
- Pulling
- Pushing

Manipulation path planning

Now it's not the (active) robot that needs to reach a destination, but a (passive) object. Many forms of manipulation have been studied:

- Grasping
- Squeezing
- Throwing
- Pulling
- Pushing

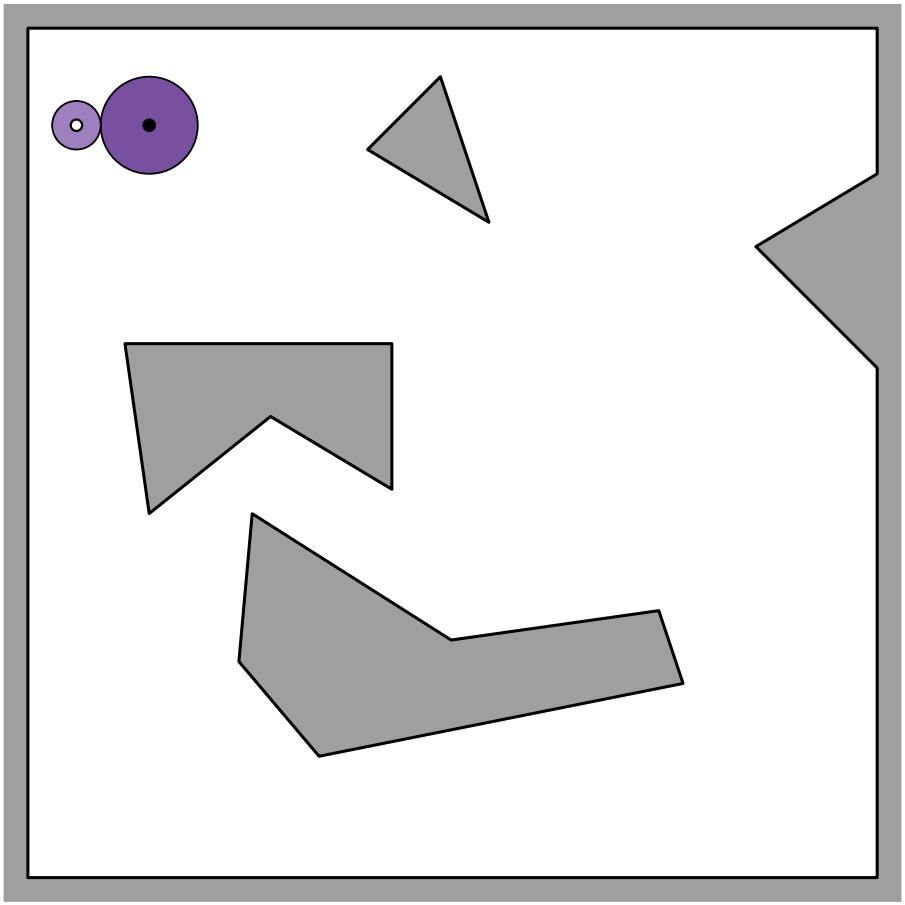
The global problem



Given:

- Two circular disks in the plane: a smaller pusher and a larger object.

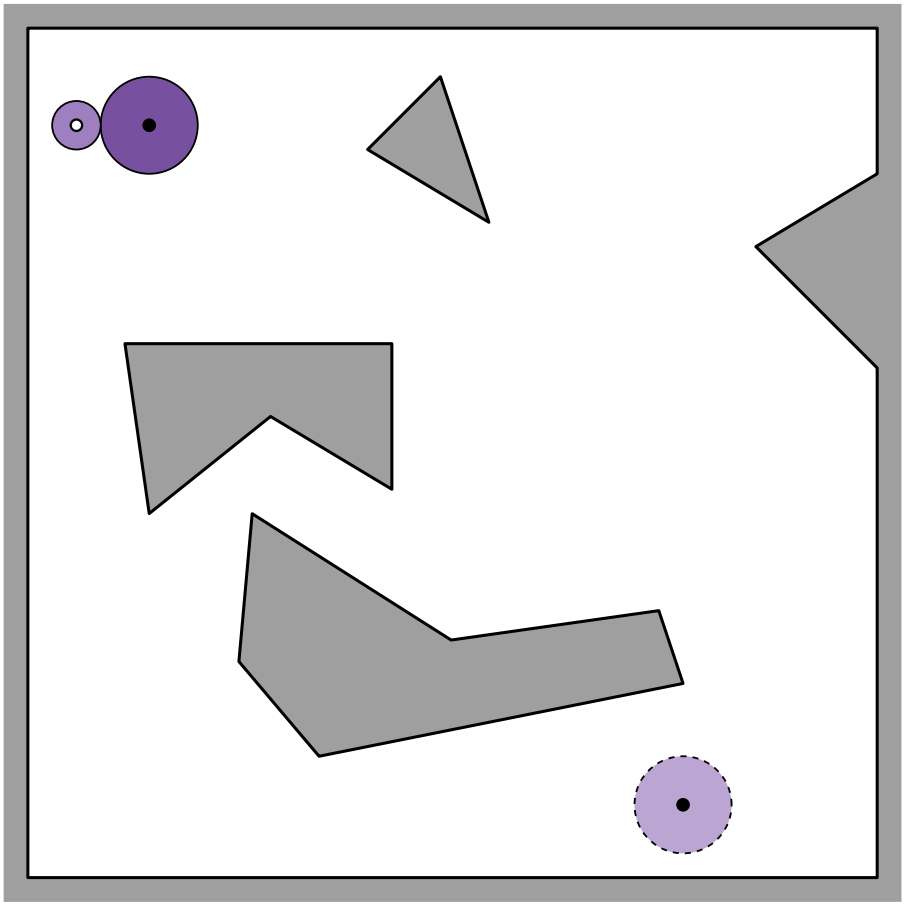
The global problem



Given:

- Two circular disks in the plane: a smaller pusher and a larger object.
- A collection of line-segment obstacles.

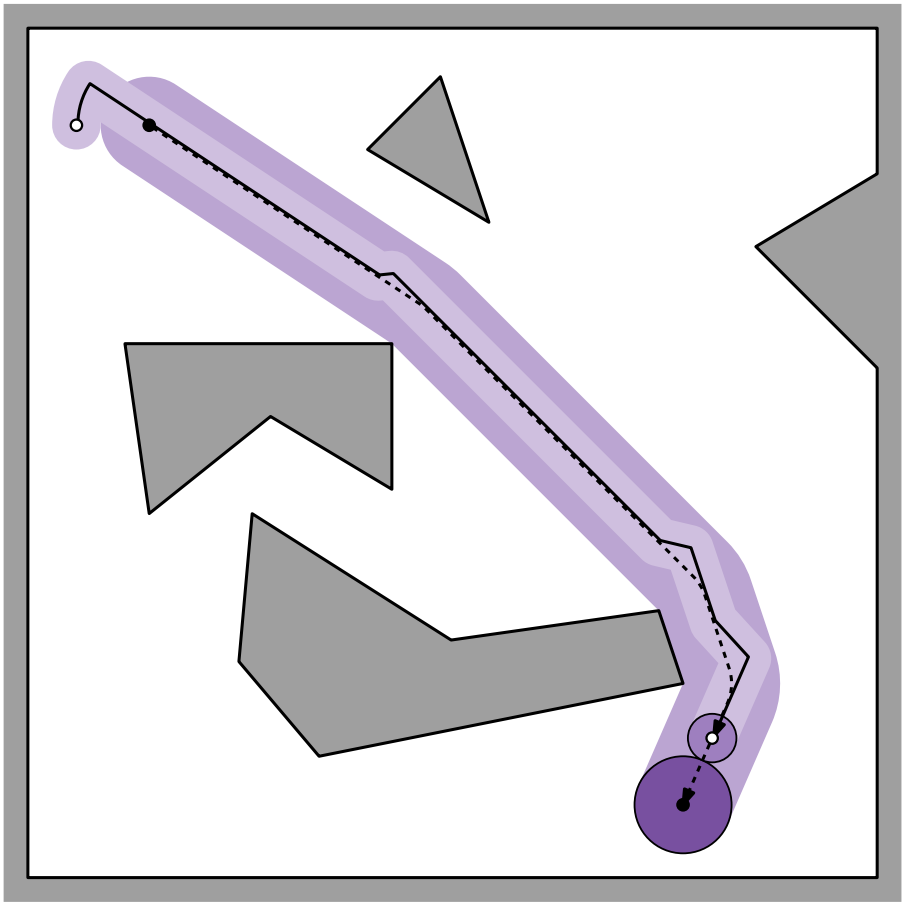
The global problem



Given:

- Two circular disks in the plane: a smaller pusher and a larger object.
- A collection of line-segment obstacles.
- A destination position for the object.

The global problem



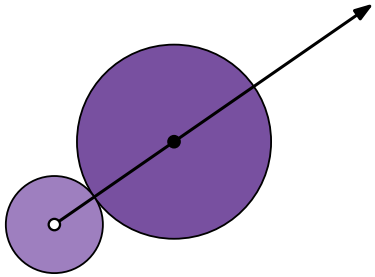
Given:

- Two circular disks in the plane: a smaller pusher and a larger object.
- A collection of line-segment obstacles.
- A destination position for the object.

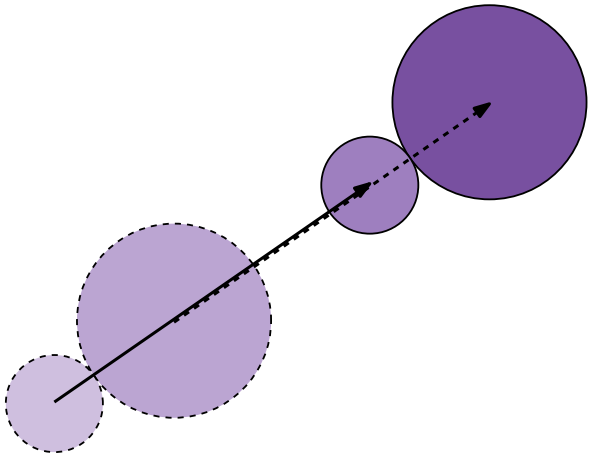
Find:

- A path for the pusher to follow (a push plan) that makes it push the object to its destination.
  (Or report that none exists.)

Compliance & path sections (1/2)

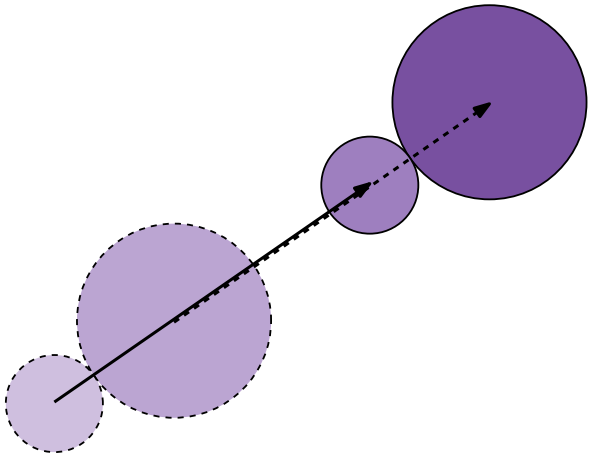


Compliance & path sections (1/2)

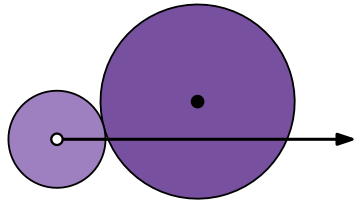


straight line

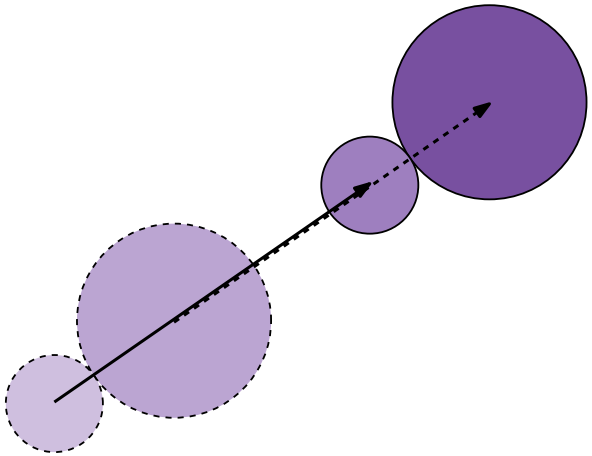
Compliance & path sections (1/2)



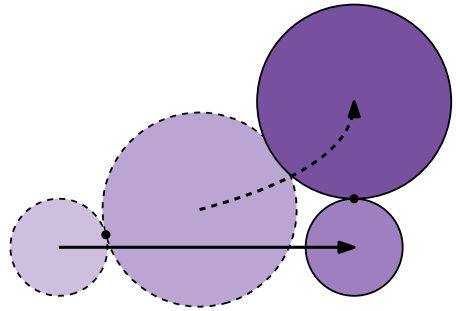
straight line



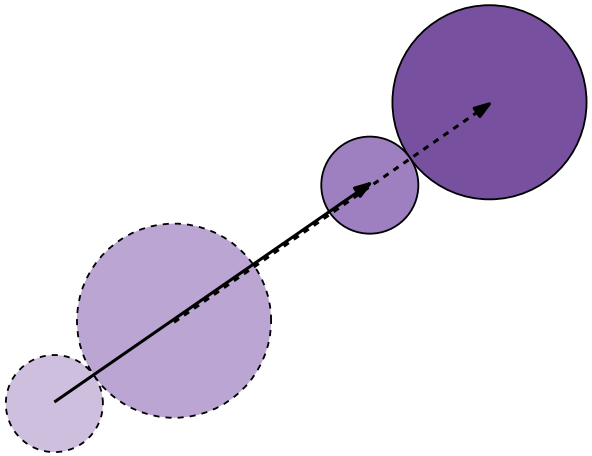
Compliance & path sections (1/2)



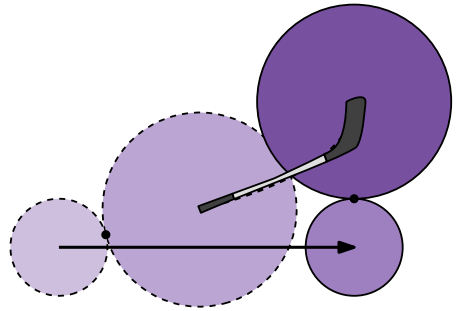
straight line



Compliance & path sections (1/2)

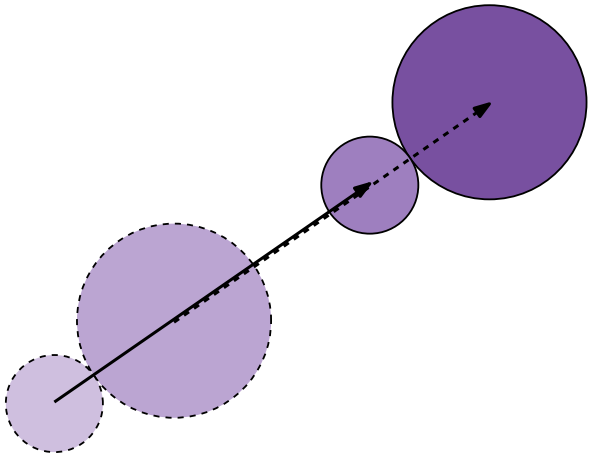


straight line

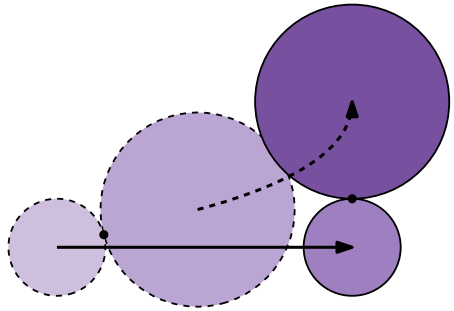


hockey stick

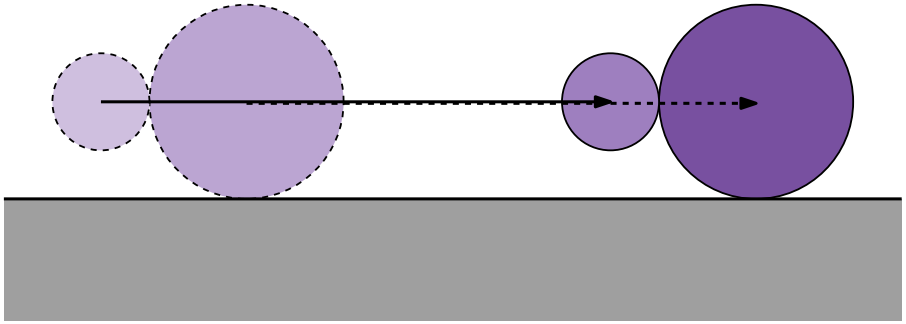
Compliance & path sections (1/2)



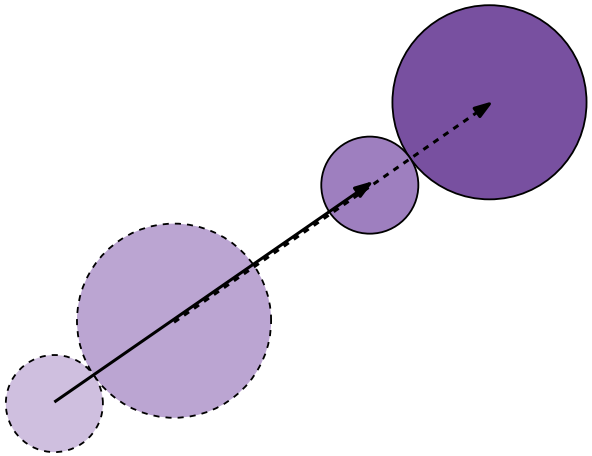
straight line



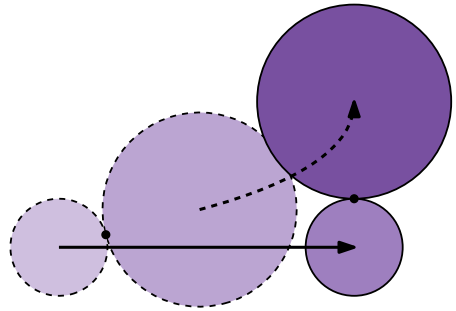
hockey stick



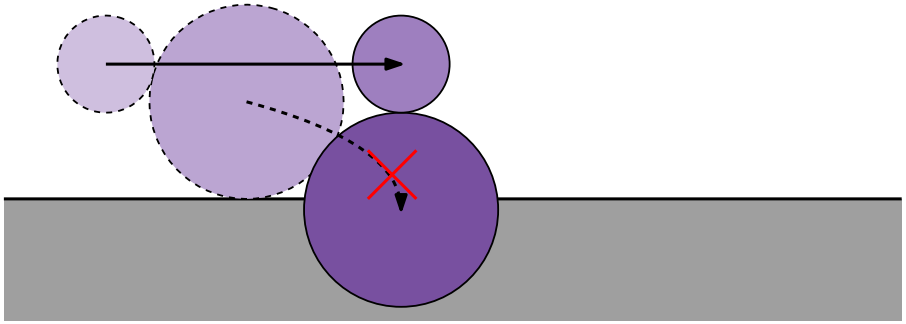
Compliance & path sections (1/2)



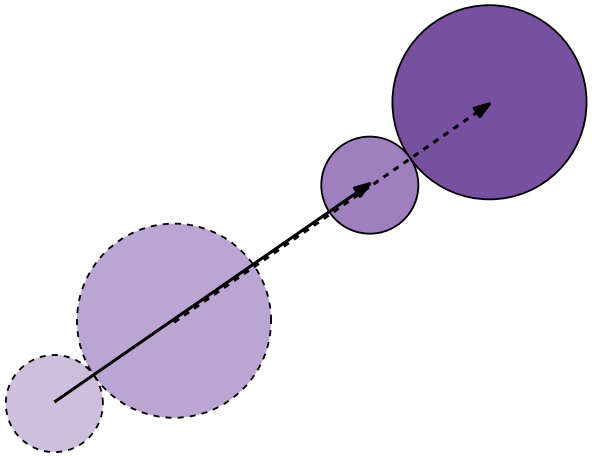
straight line



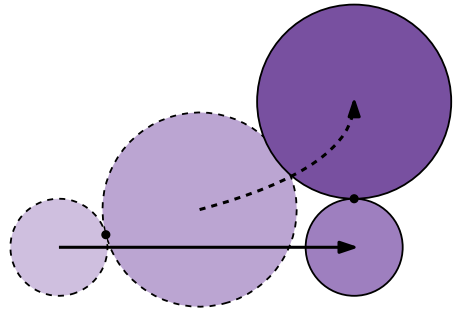
hockey stick



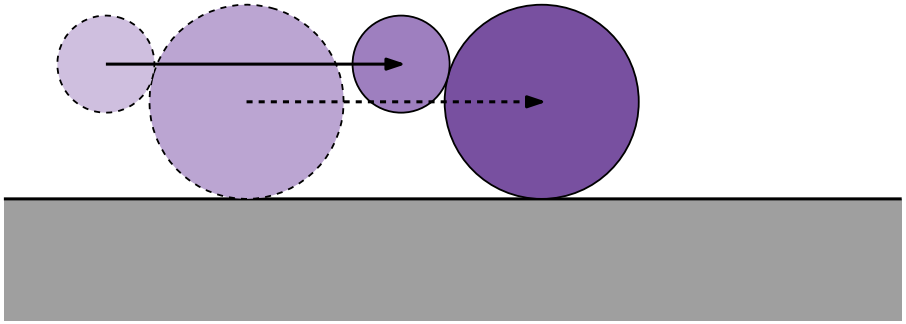
Compliance & path sections (1/2)



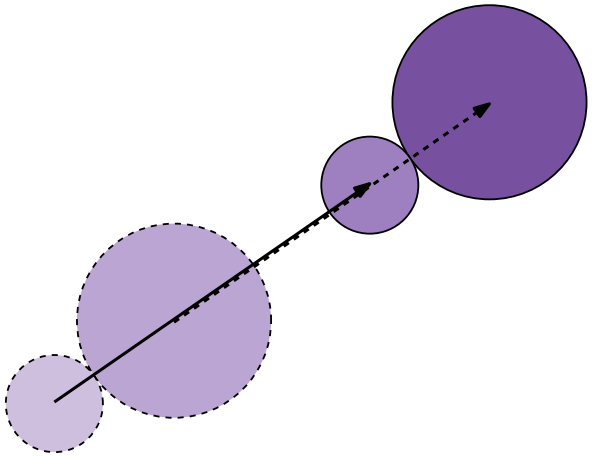
straight line



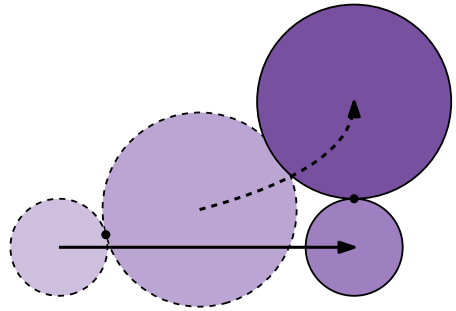
hockey stick



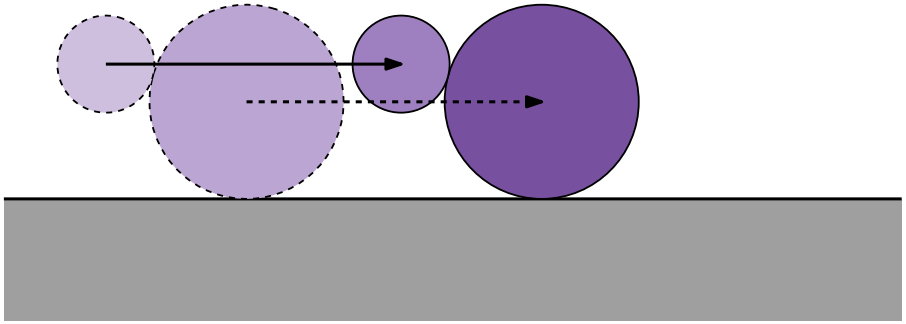
Compliance & path sections (1/2)



straight-line non-compliant

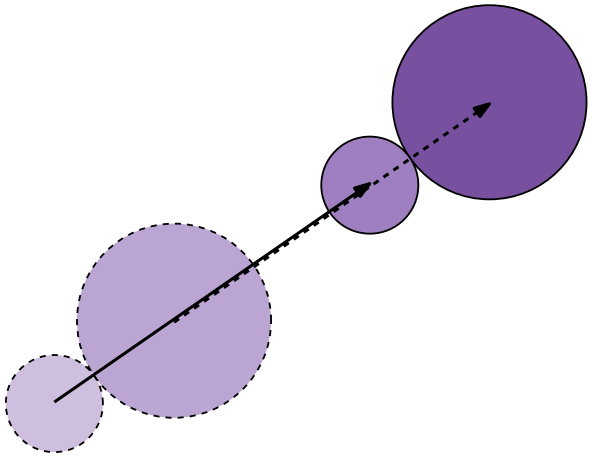


hockey-stick non-compliant

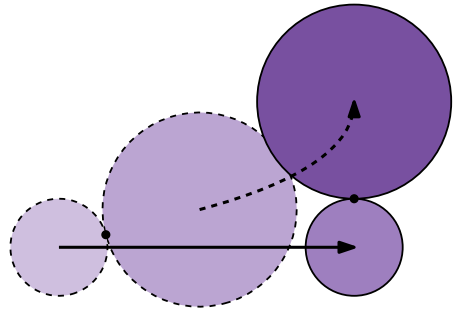


straight-line compliant

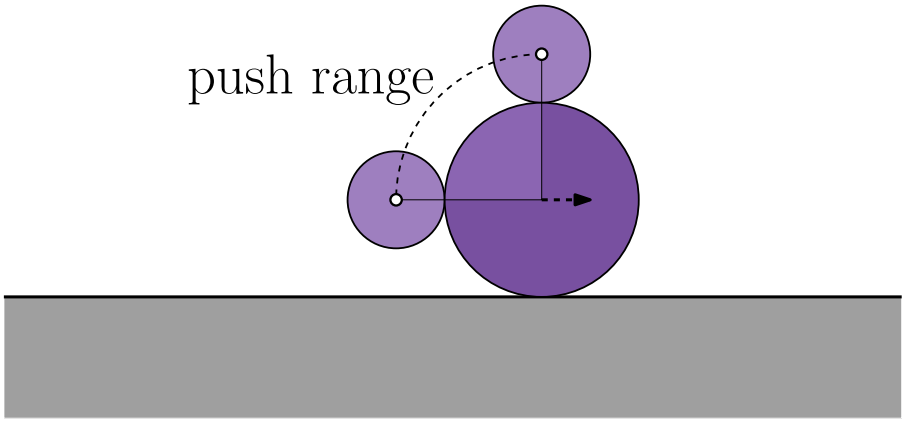
Compliance & path sections (1/2)



straight-line non-compliant

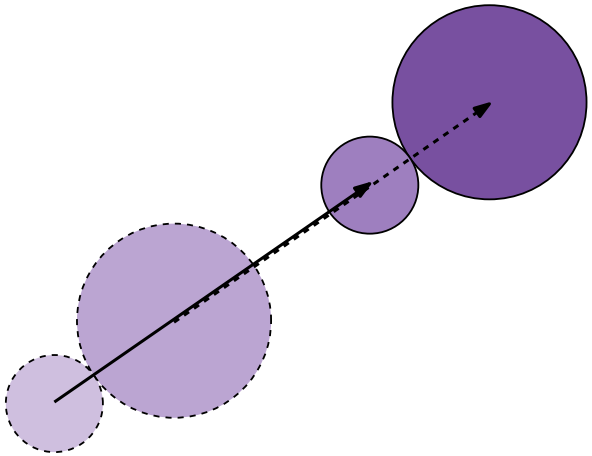


hockey-stick non-compliant

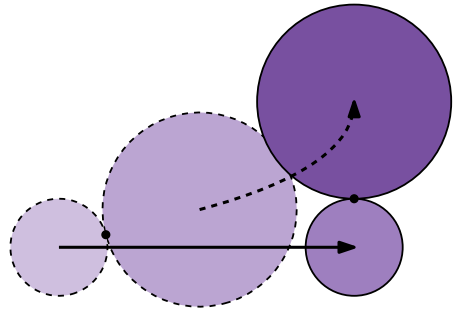


straight-line compliant

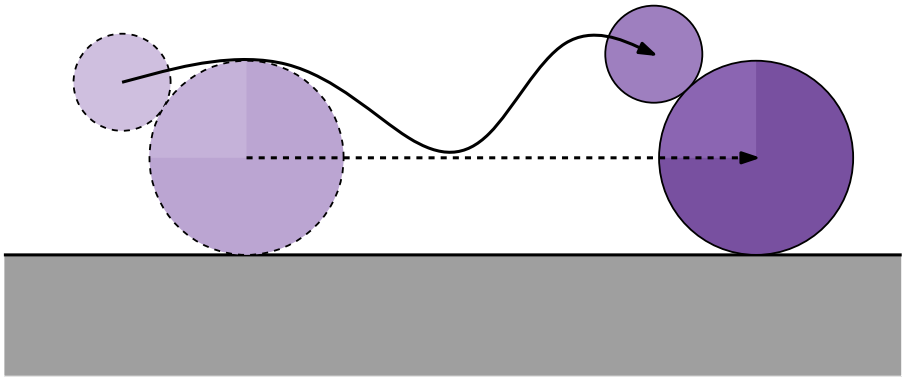
Compliance & path sections (1/2)



straight-line non-compliant

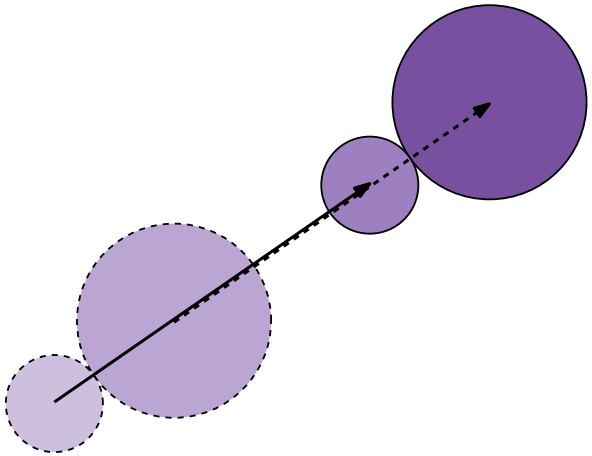


hockey-stick non-compliant

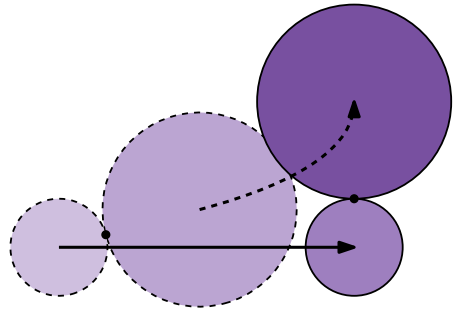


straight-line compliant

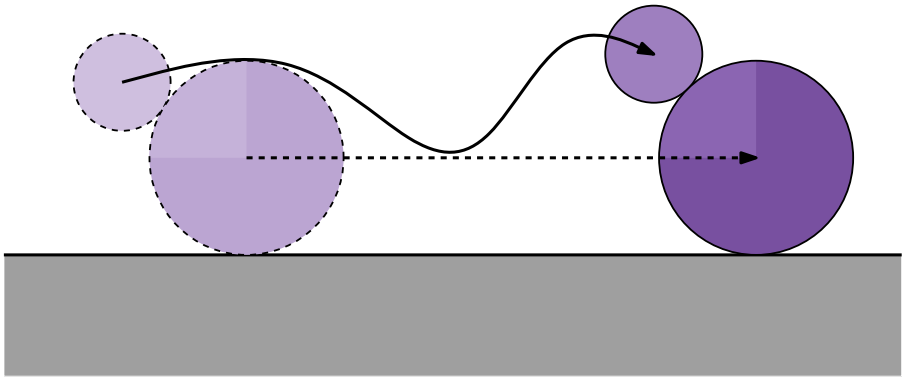
Compliance & path sections (1/2)



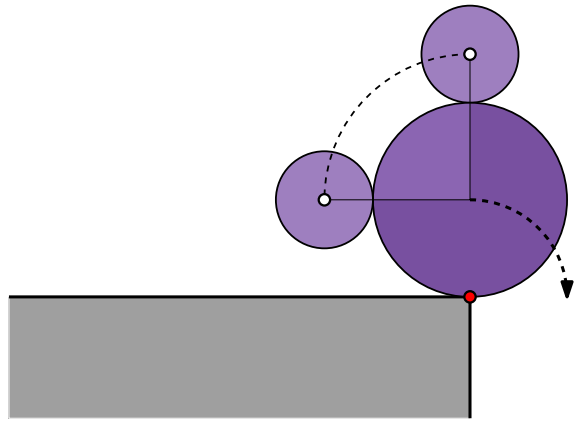
straight-line non-compliant



hockey-stick non-compliant

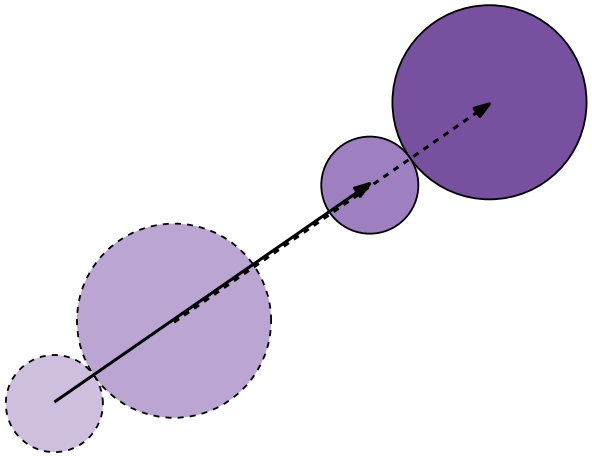


straight-line compliant

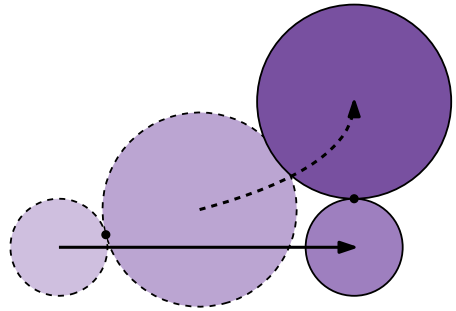


circular compliant

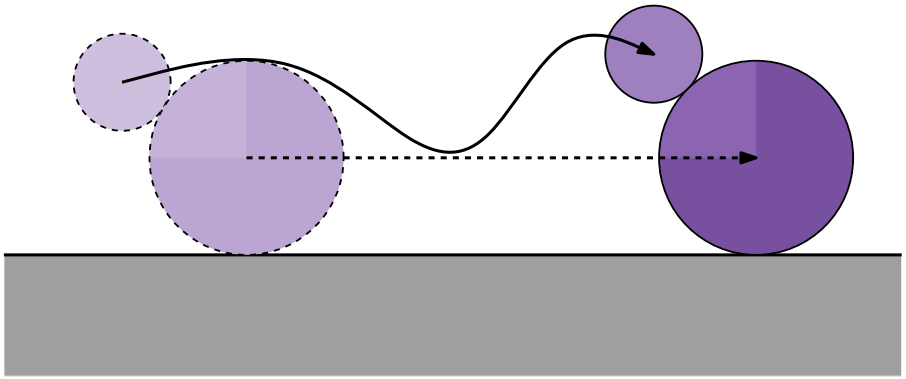
Compliance & path sections (1/2)



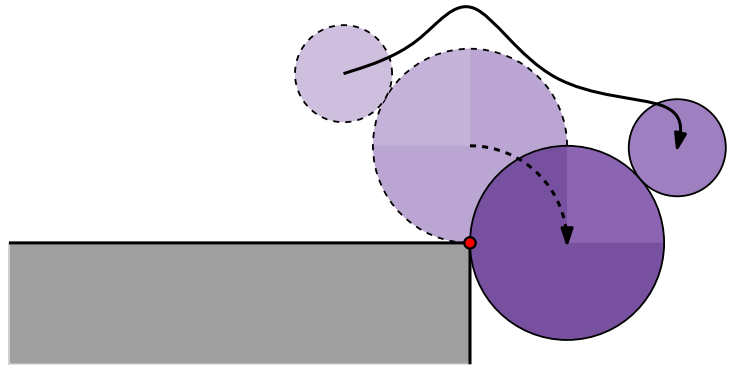
straight-line non-compliant



hockey-stick non-compliant

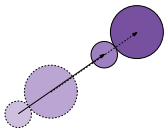


straight-line compliant

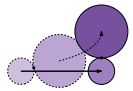


circular compliant

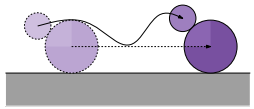
Compliance & path sections (2/2)



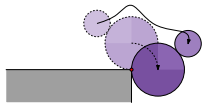
straight-line non-compliant



hockey-stick non-compliant



straight-line compliant

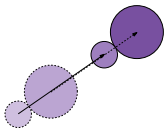


circular compliant

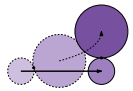
path sections

(object moves along curve, pusher stays within associated push range)

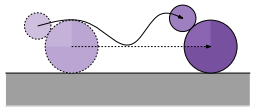
Compliance & path sections (2/2)



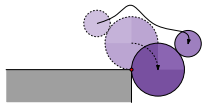
straight-line non-compliant



hockey-stick non-compliant



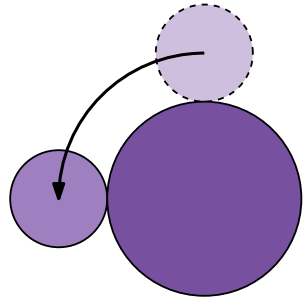
straight-line compliant

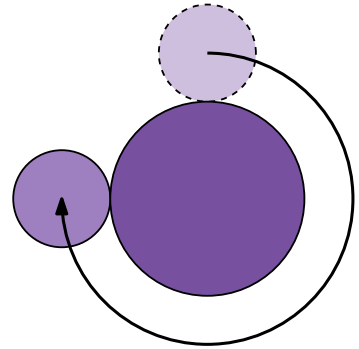


circular compliant

path sections

(object moves along curve, pusher stays within associated push range)





contact transits

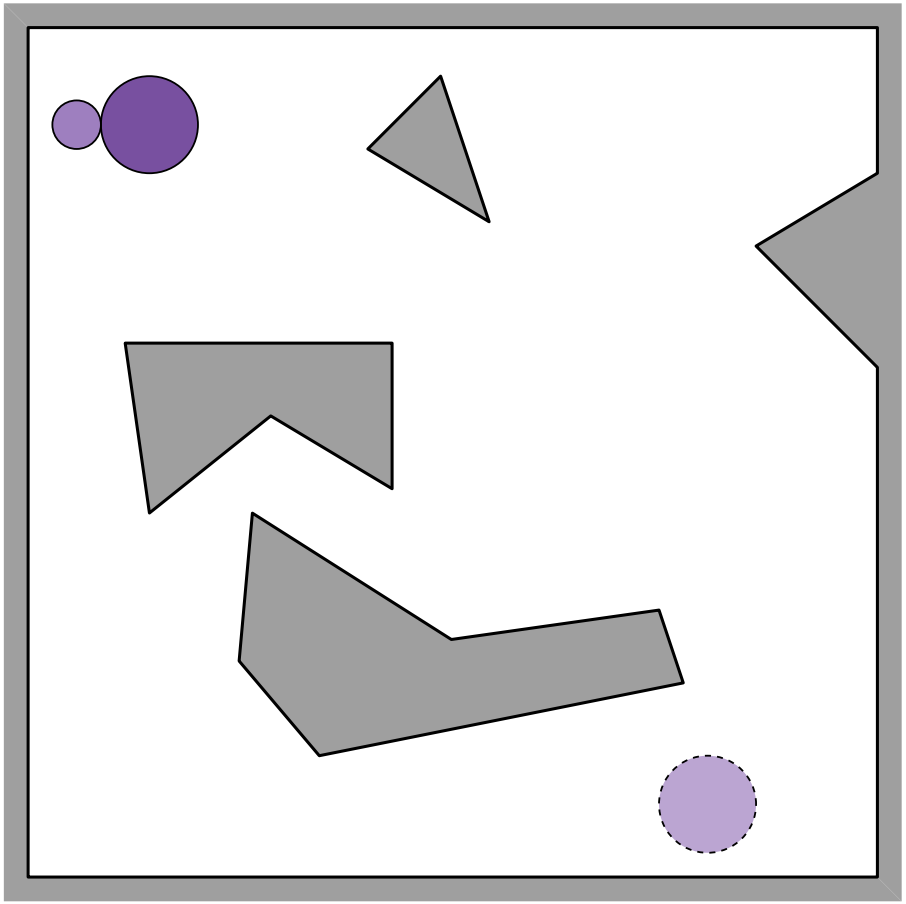
(object is stationary, pusher turns around it)

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

Nieuwenhuisen's algorithm

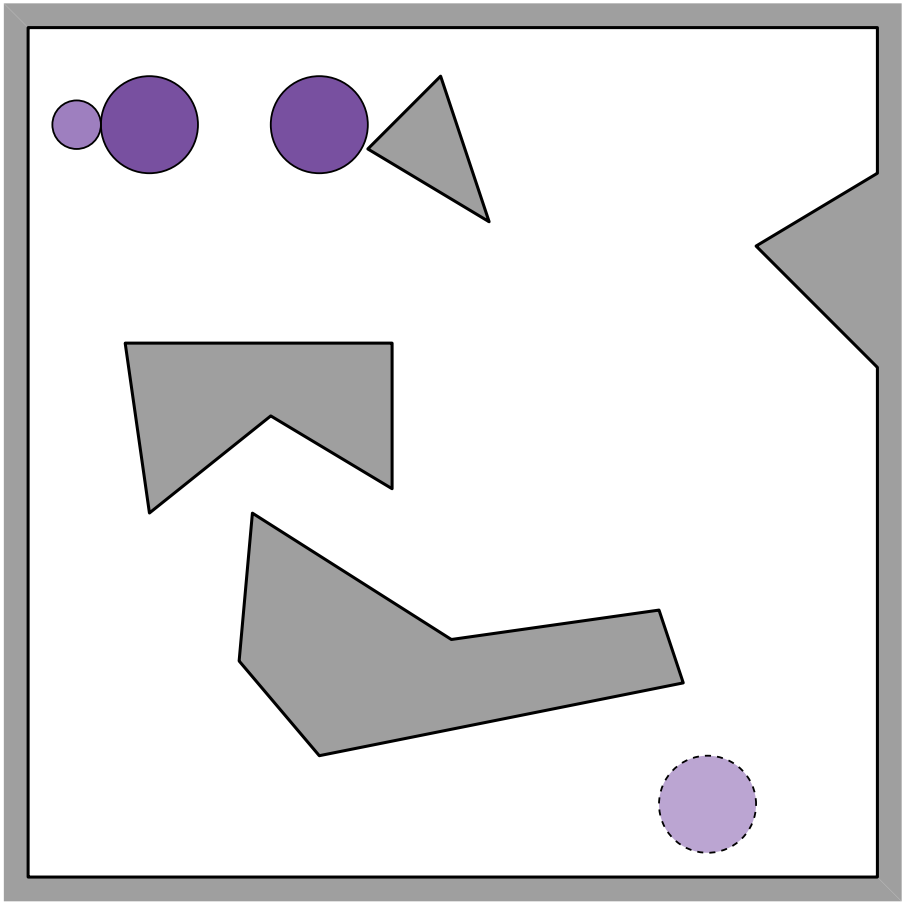
Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.



Create a tree with the initial configuration as the root:

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

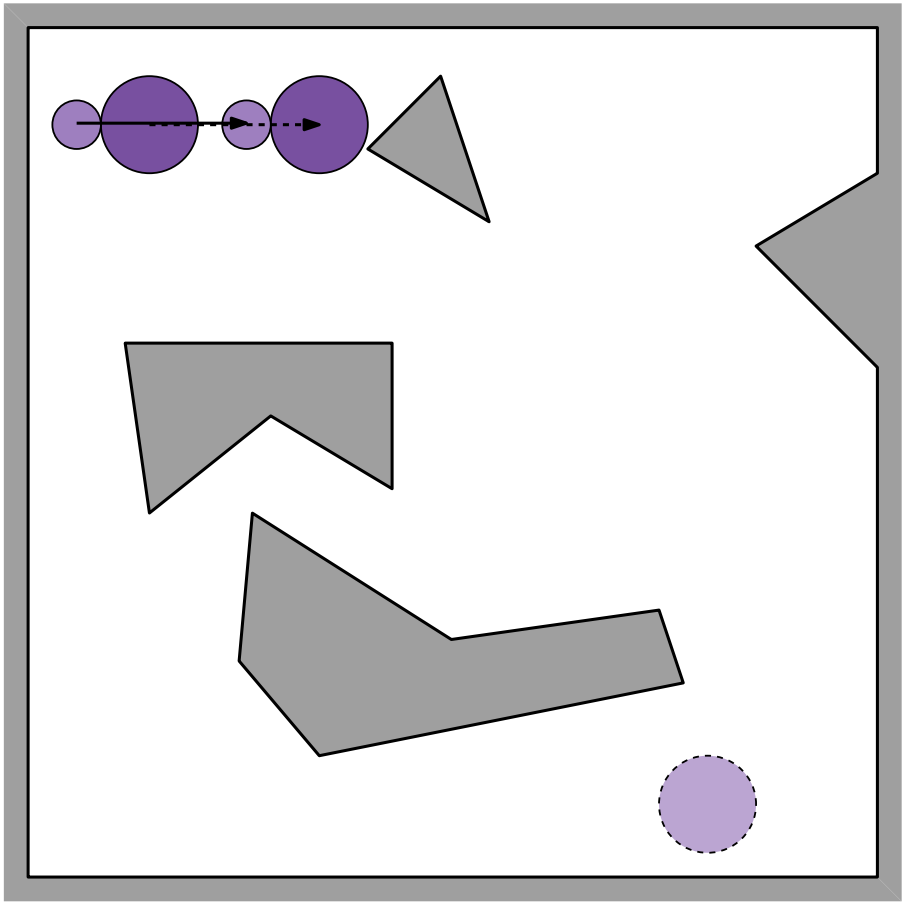


Create a tree with the initial configuration as the root:

- Take a random position.

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

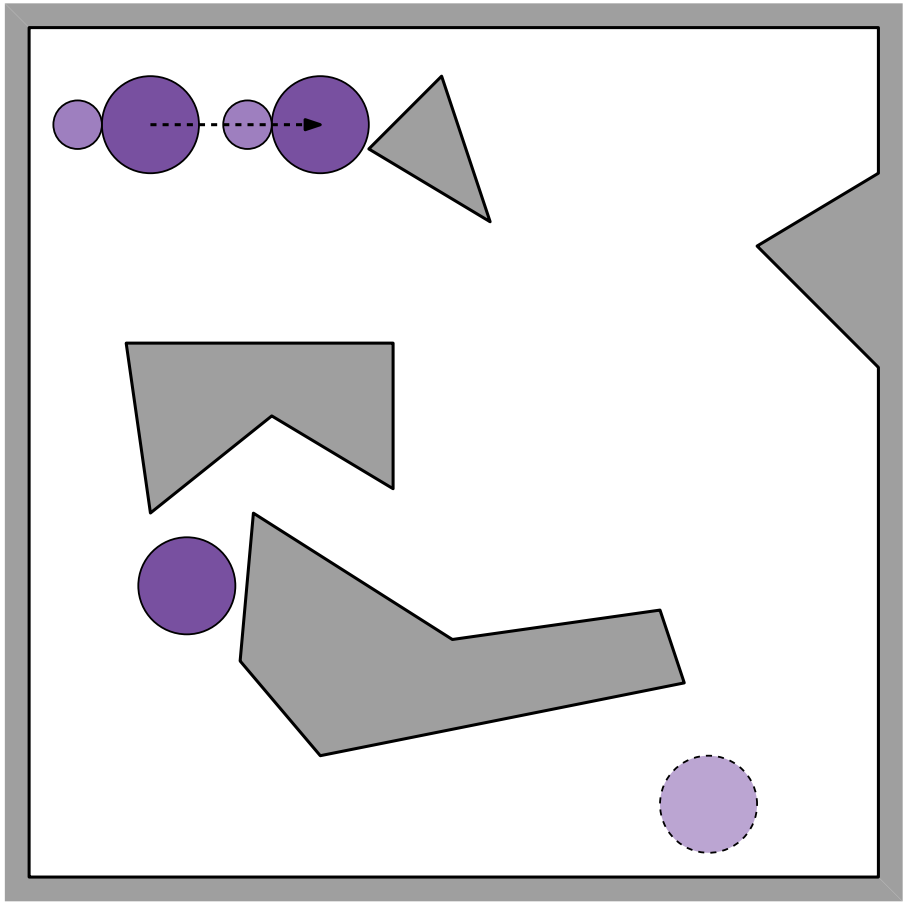


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success $\longrightarrow$ add to tree.

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

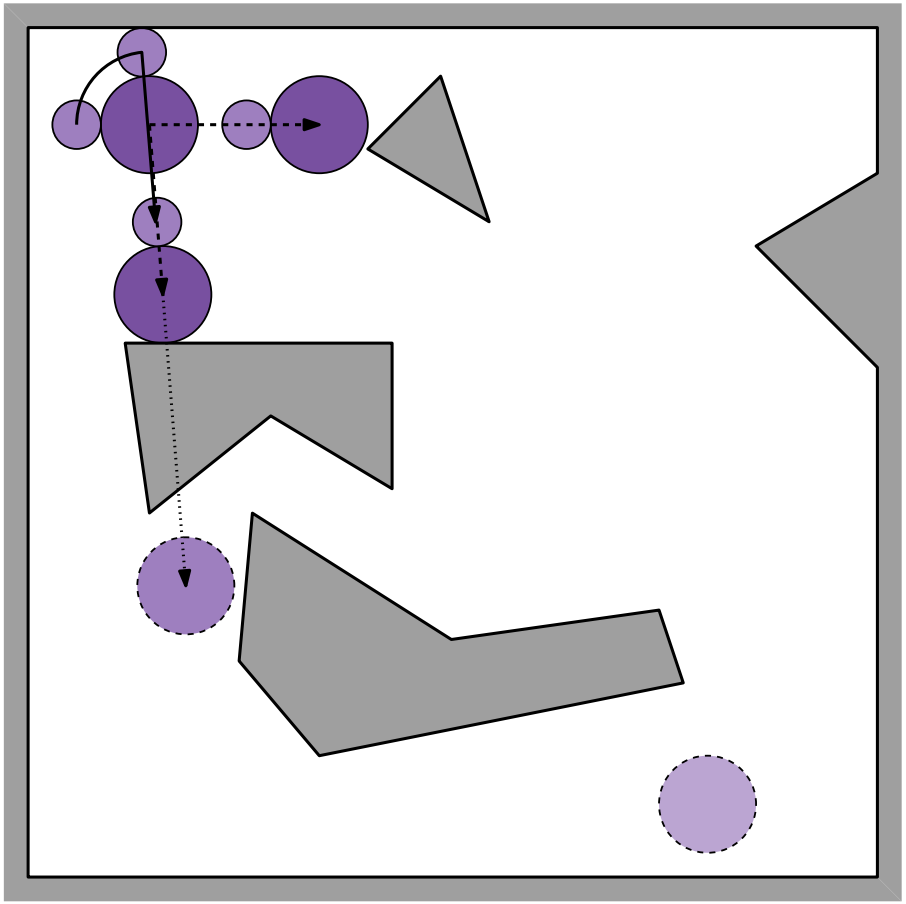


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success $\longrightarrow$ add to tree.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

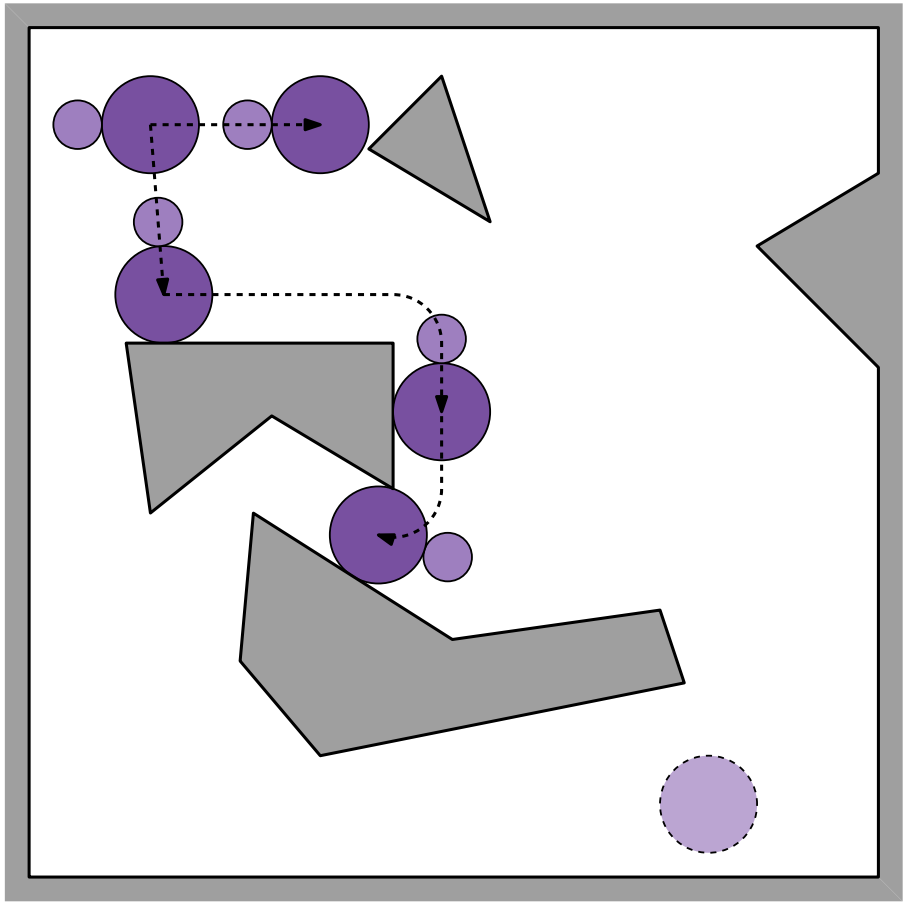


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

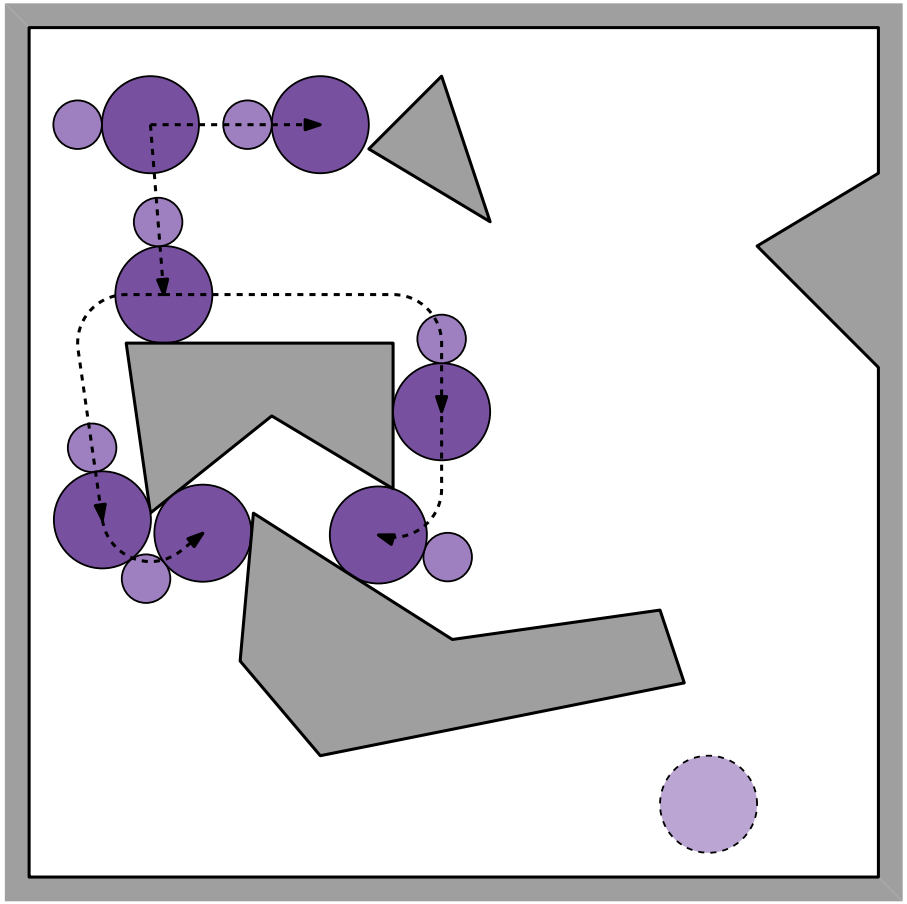


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

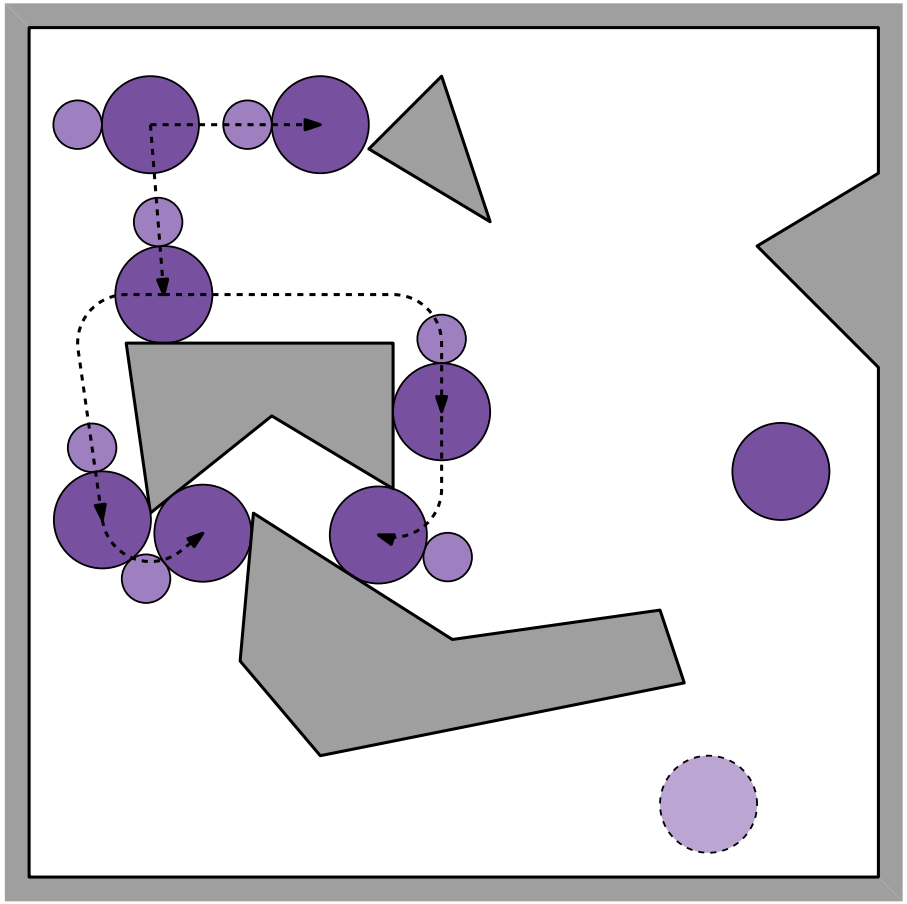


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

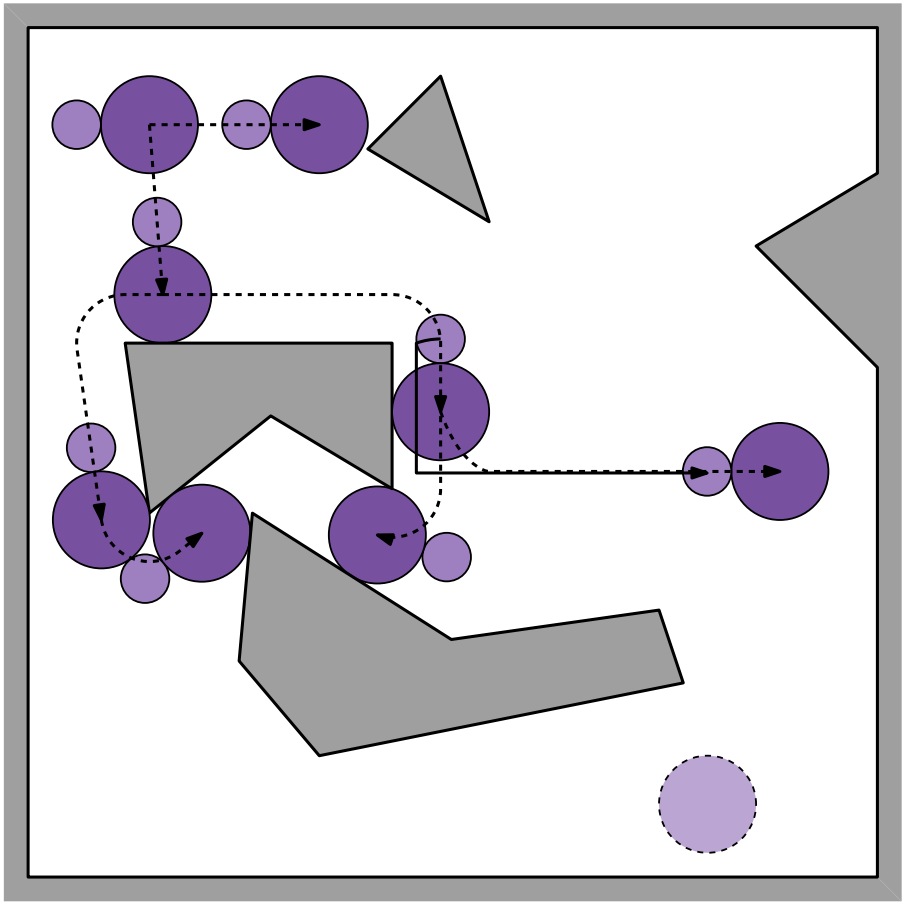


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

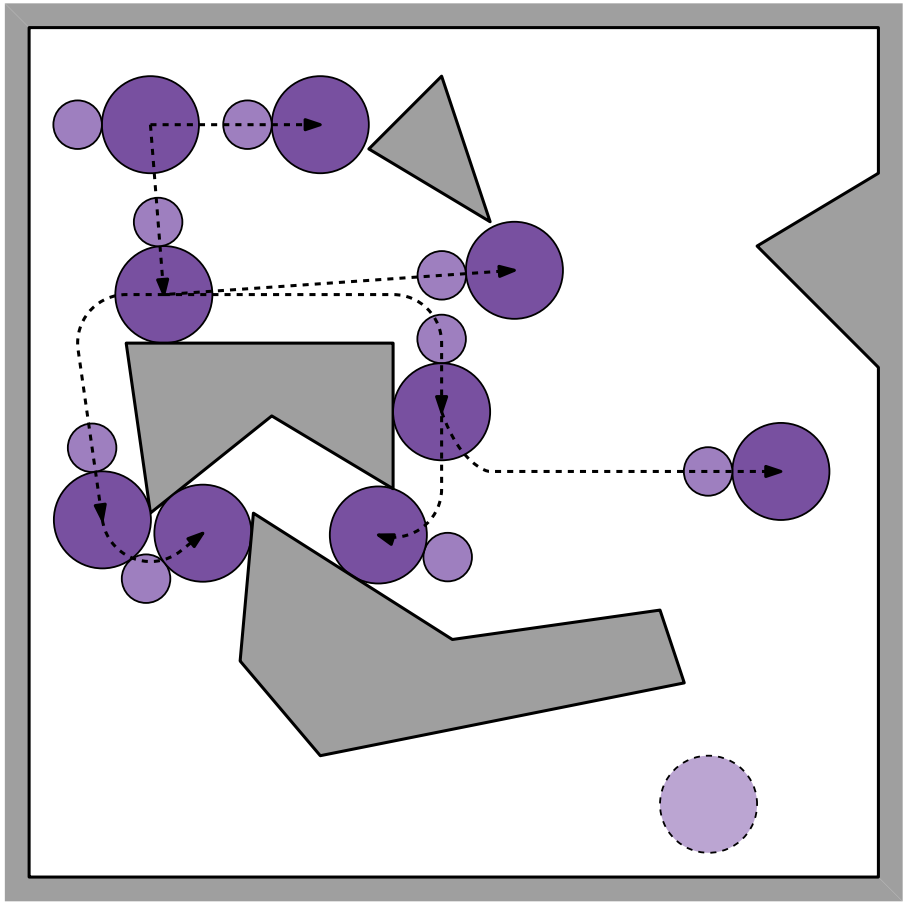


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

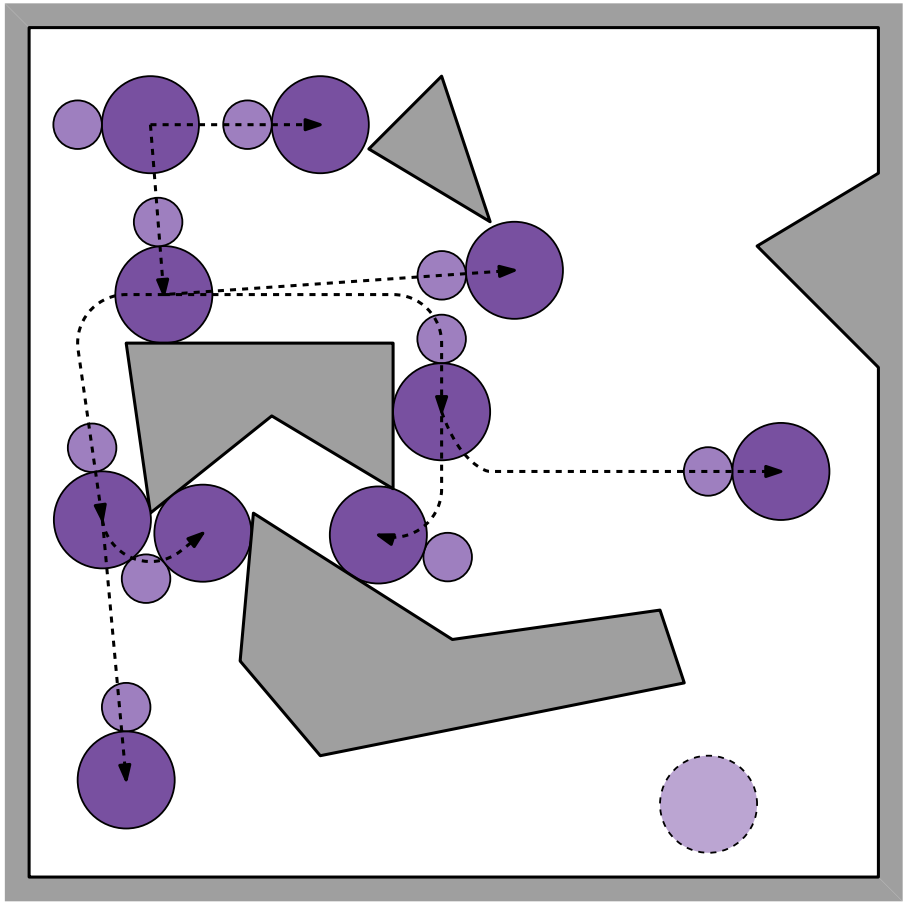


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

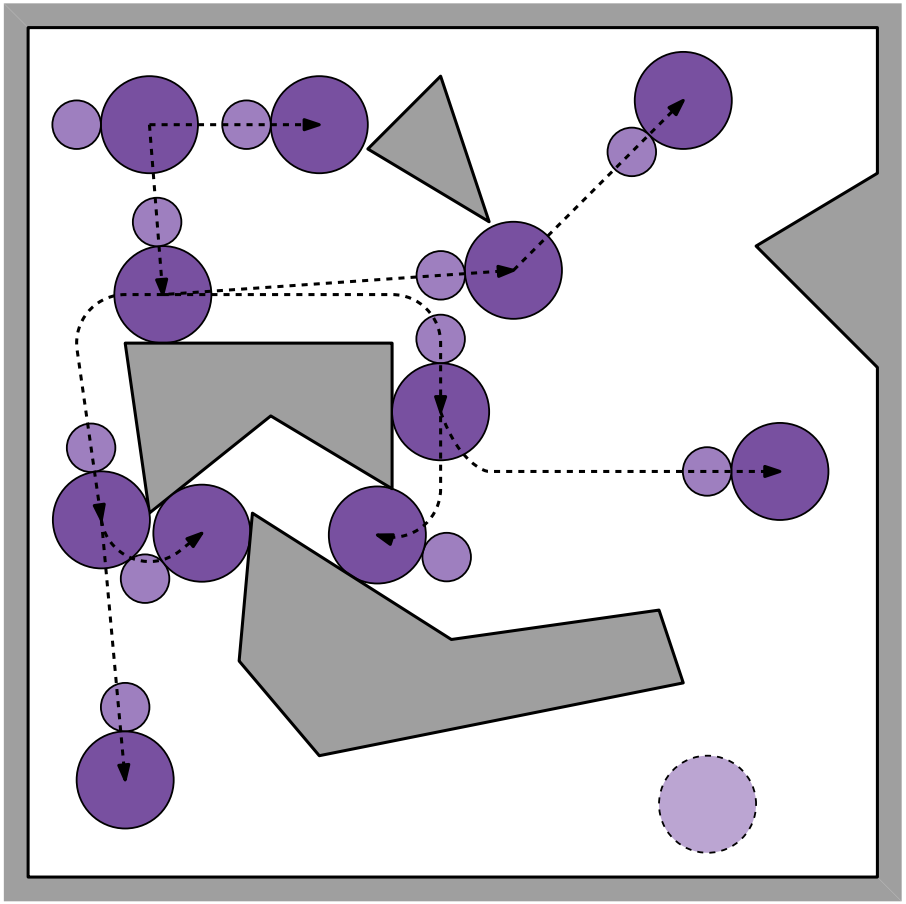


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

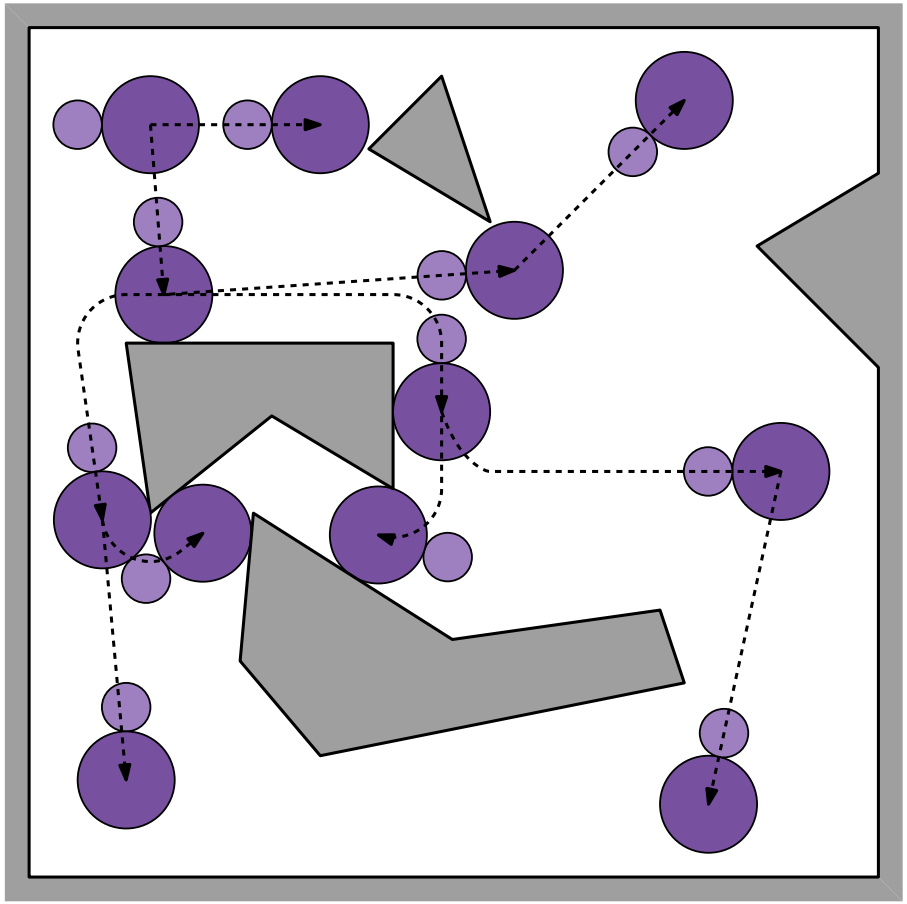


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

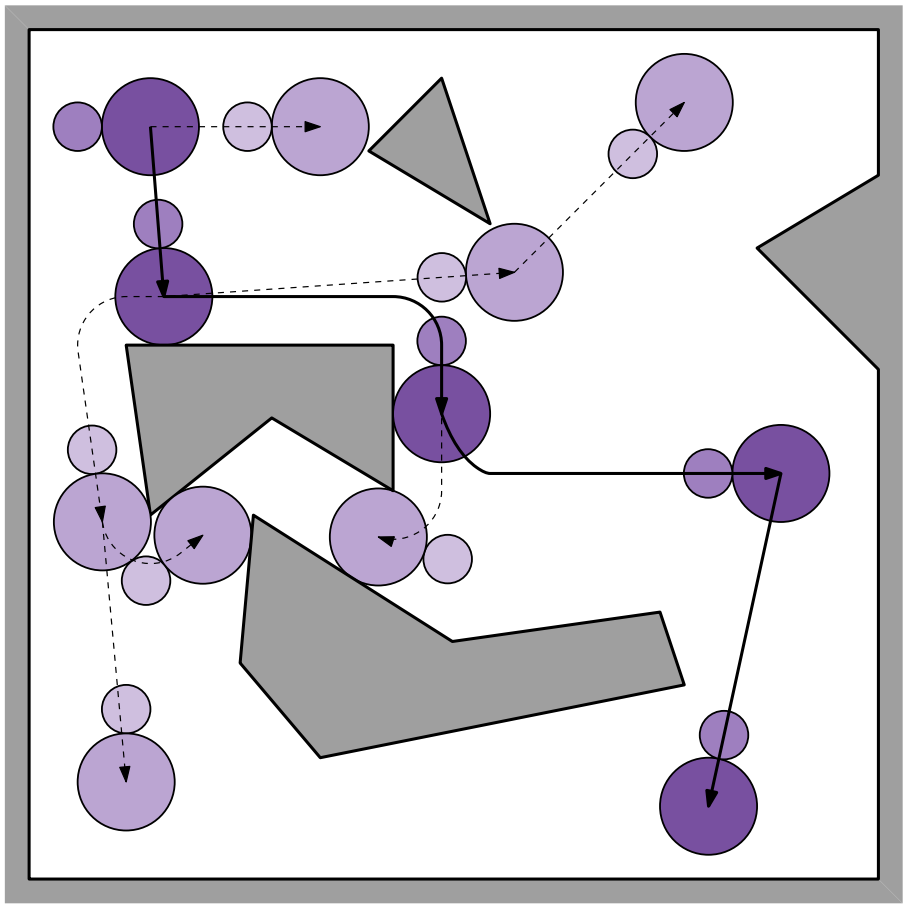


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat until destination gets added to the tree (or other stopping criterion).

Nieuwenhuisen's algorithm

Dennis Nieuwenhuisen designed an algorithm for this problem based on the Rapidly-exploring Random Trees path-planning algorithm.

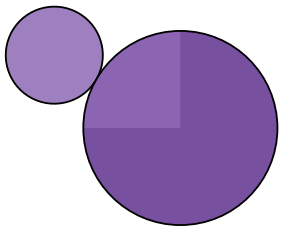


Create a tree with the initial configuration as the root:

- Take a random position.
- Try a straight line to it from the closest position already in the tree.
  Success → add to tree.
  Failure → add reachable compliant positions.
- Repeat until destination gets added to the tree (or other stopping criterion).

The studied subproblem

Nieuwenhuisen's algorithm needs a subroutine for this subproblem:

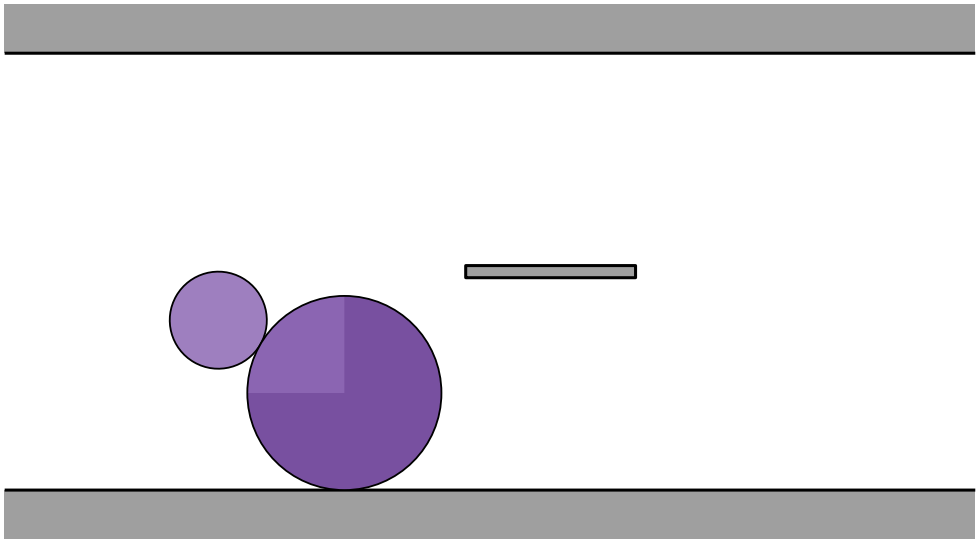


Given:

- Two circular disks in the plane: a smaller pusher and a larger object.

The studied subproblem

Nieuwenhuisen's algorithm needs a subroutine for this subproblem:

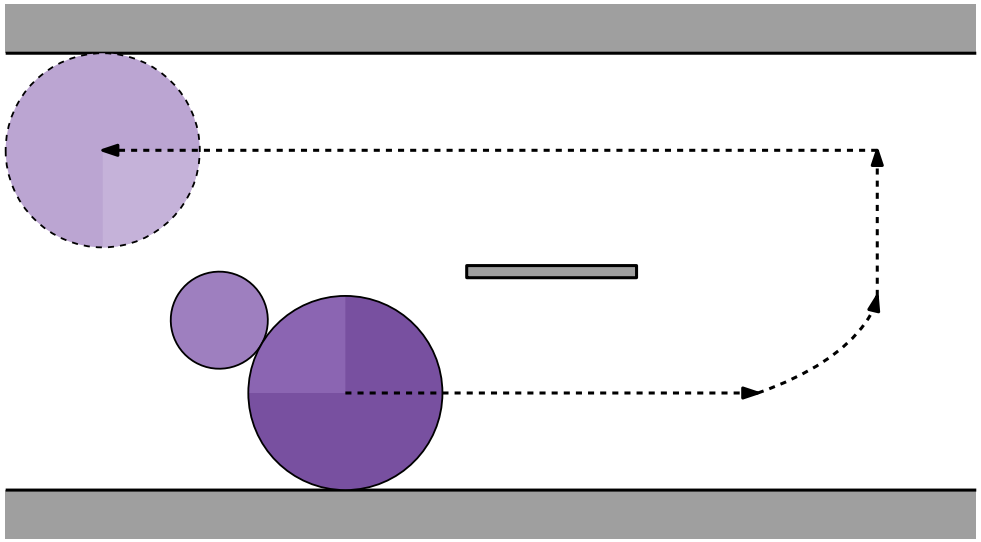


Given:

- Two circular disks in the plane: a smaller pusher and a larger object.
- A collection of line-segment obstacles.

The studied subproblem

Nieuwenhuisen's algorithm needs a subroutine for this subproblem:

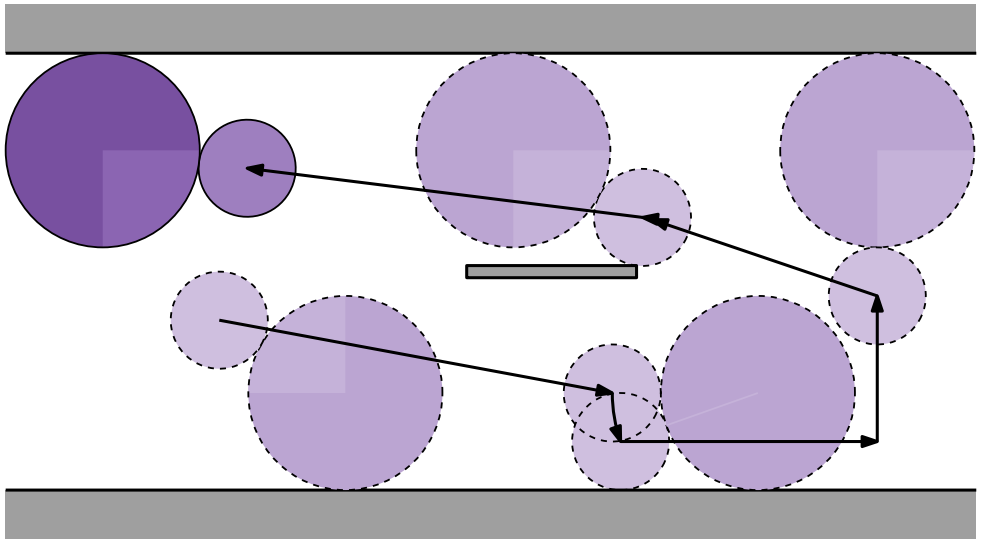


Given:

- Two circular disks in the plane: a smaller pusher and a larger object.
- A collection of line-segment obstacles.
- A path of obstacle-free path sections for the object.

The studied subproblem

Nieuwenhuisen's algorithm needs a subroutine for this subproblem:



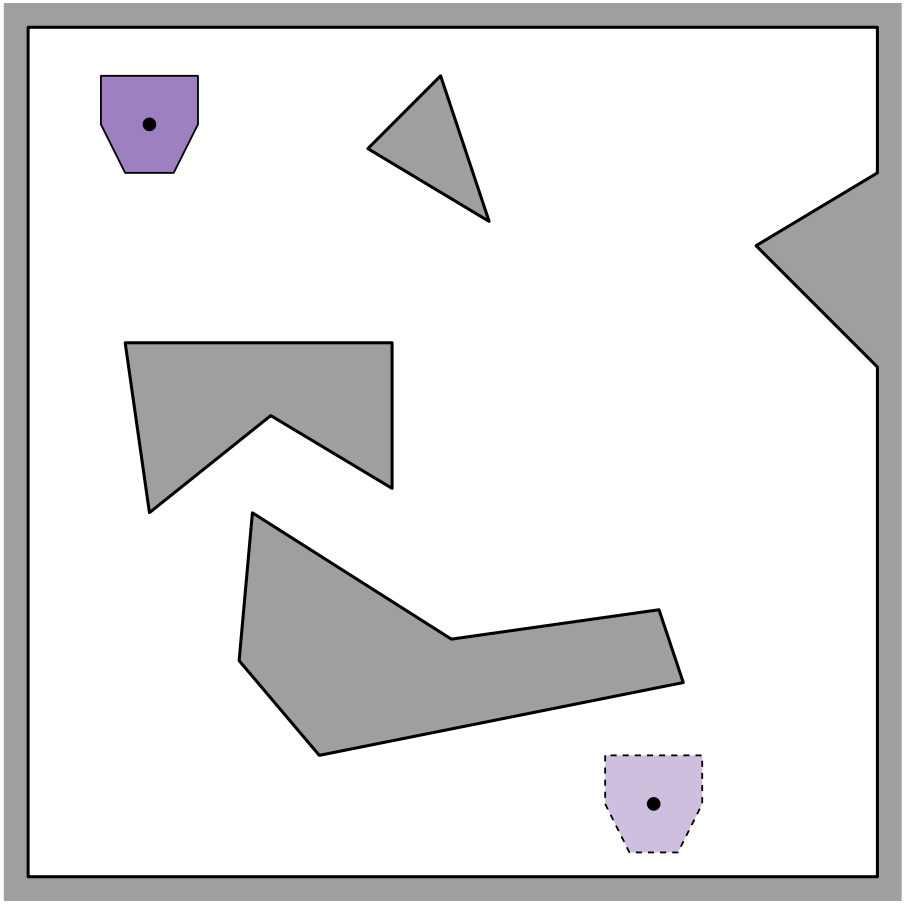
Given:

- Two circular disks in the plane: a smaller pusher and a larger object.
- A collection of line-segment obstacles.
- A path of obstacle-free path sections for the object.

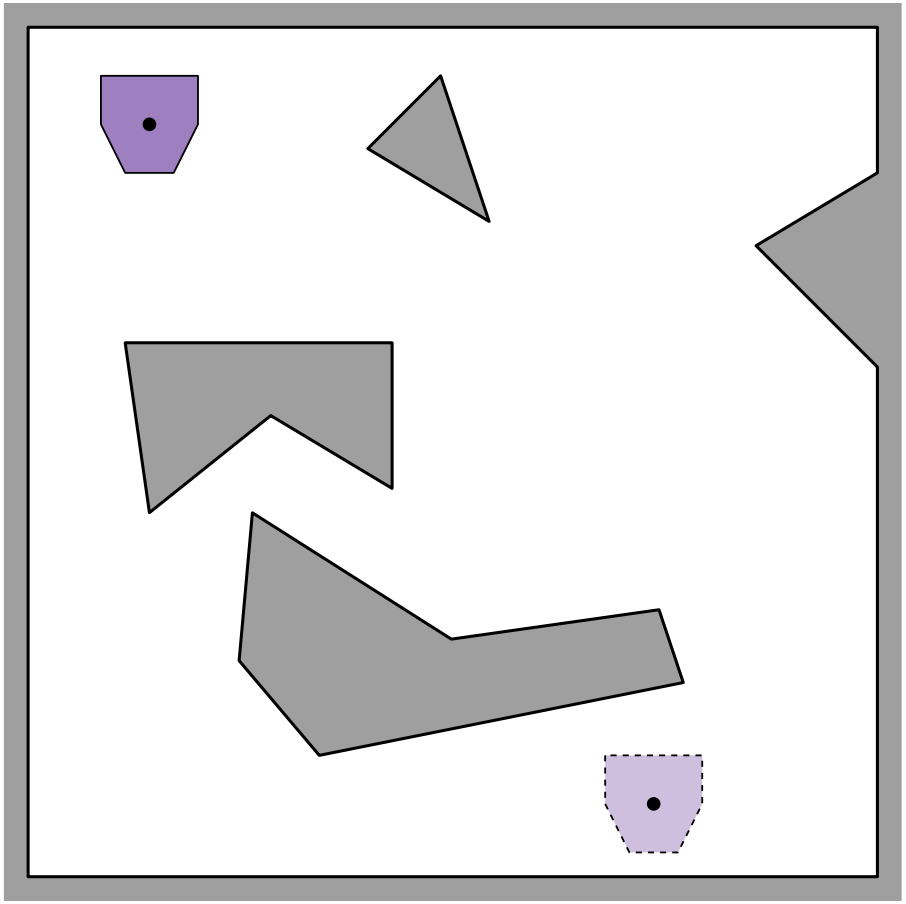
Find:

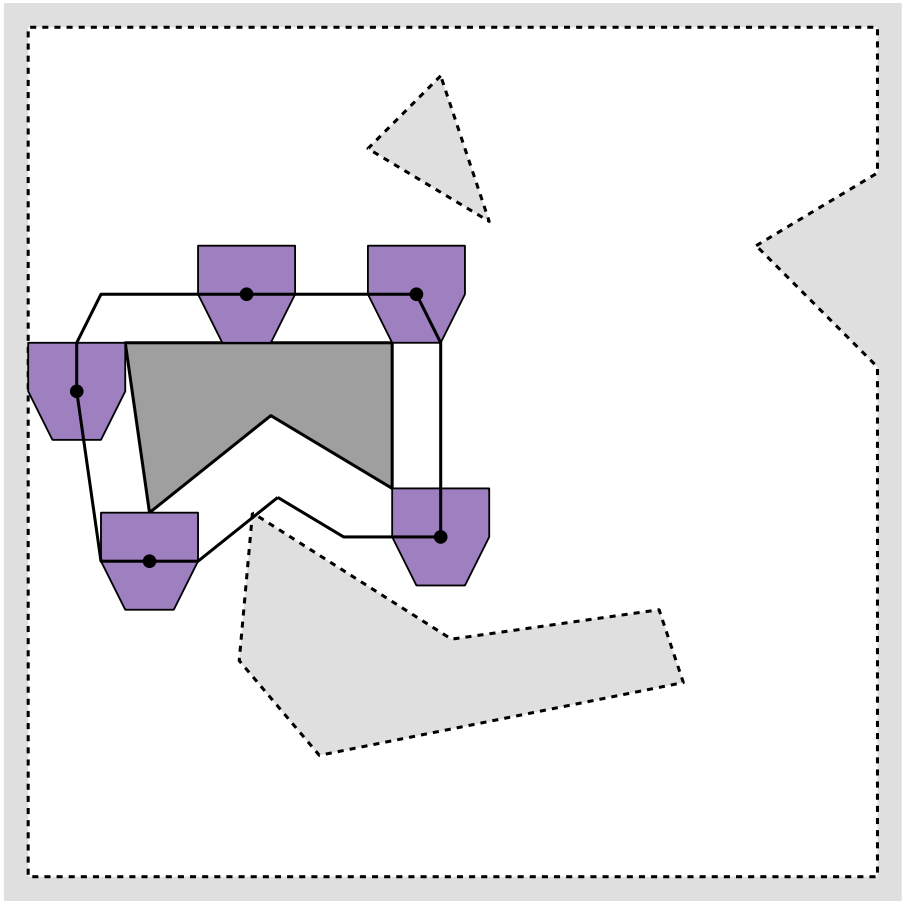
- A push plan that makes the pusher push the object (as far as possible) along its path.

The configuration-space method

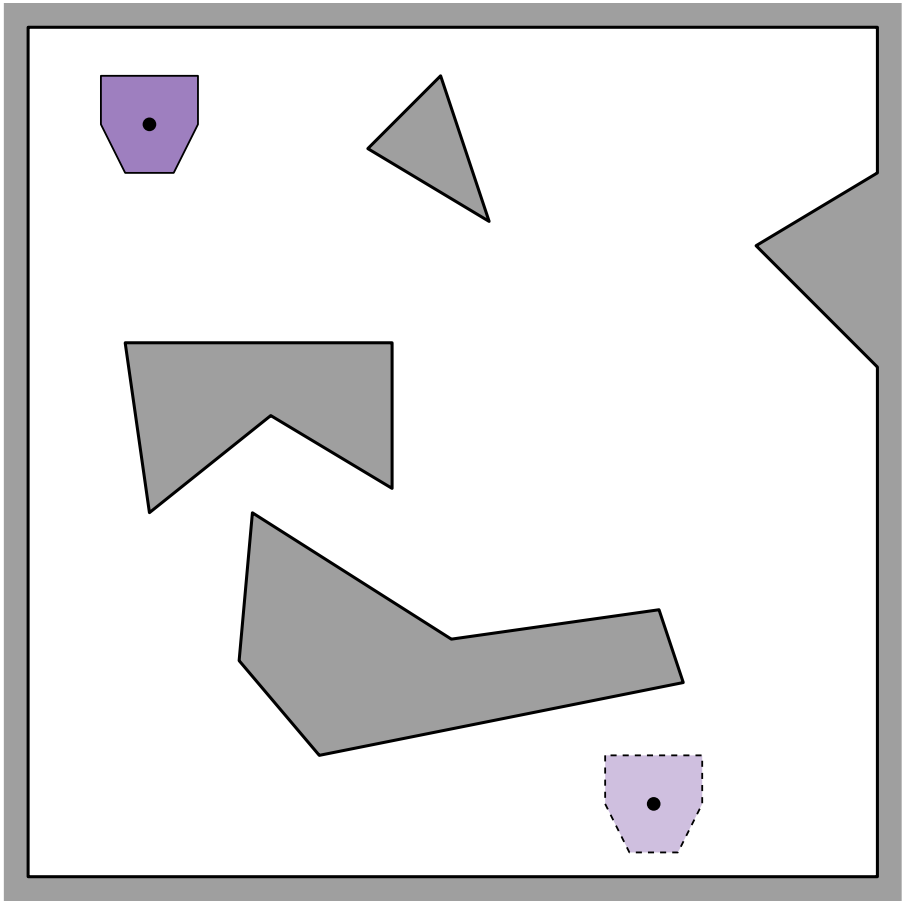


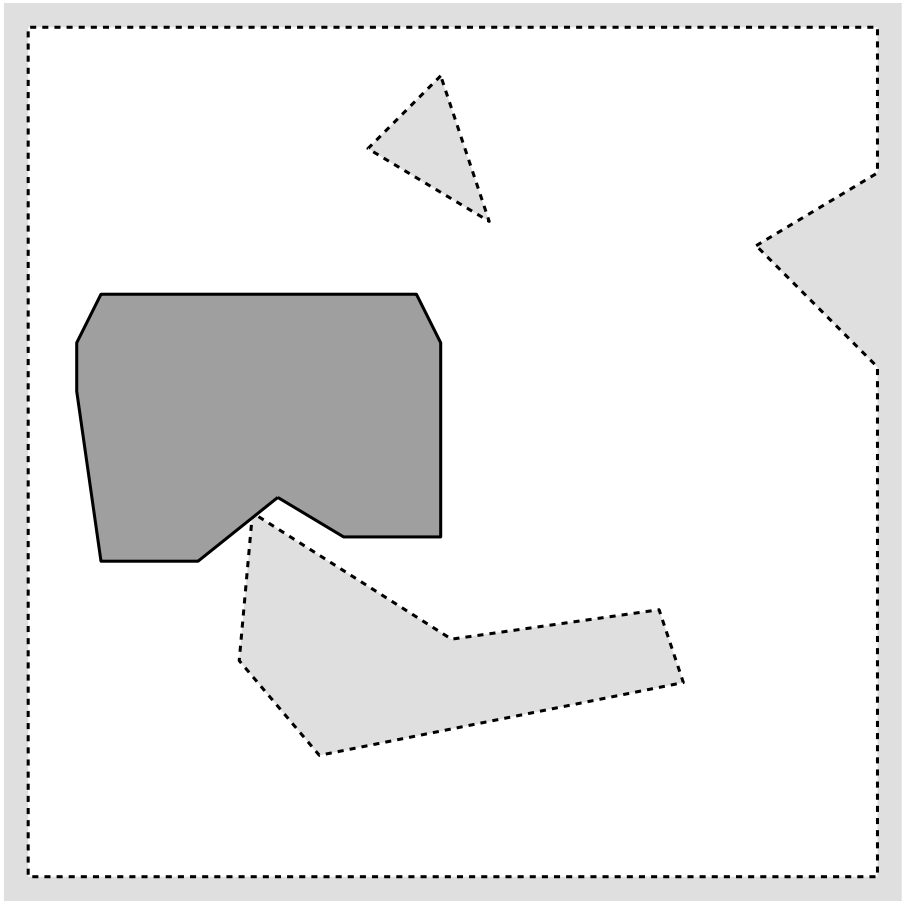
The configuration-space method



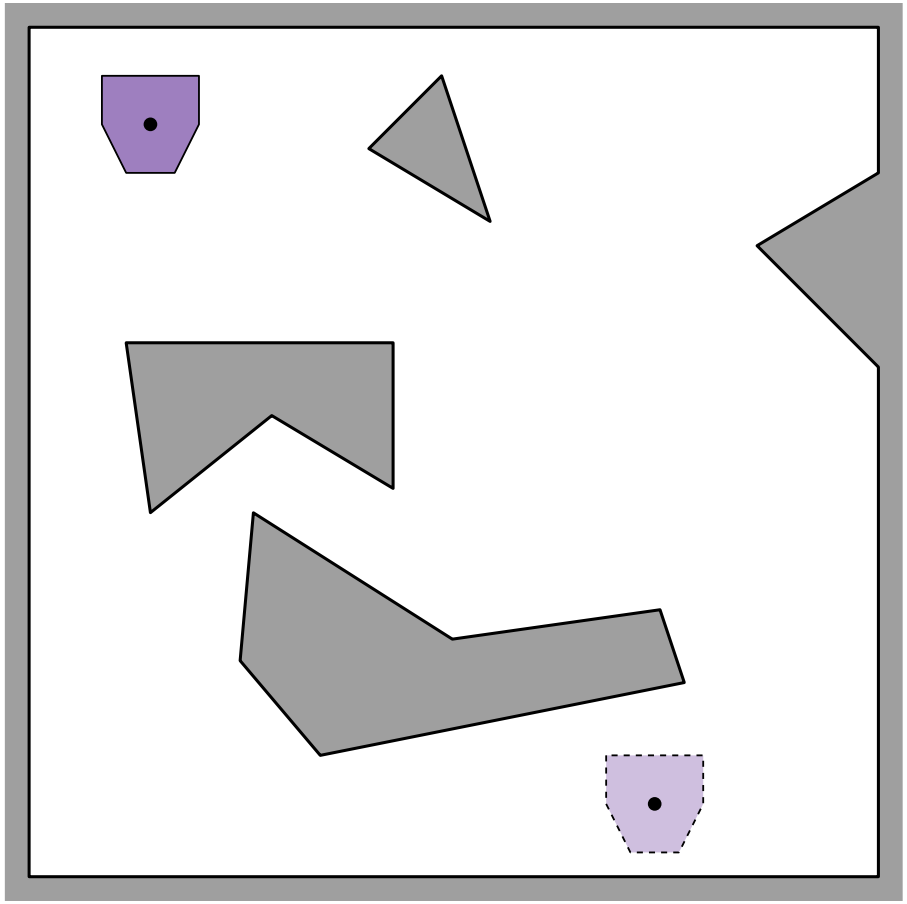


The configuration-space method

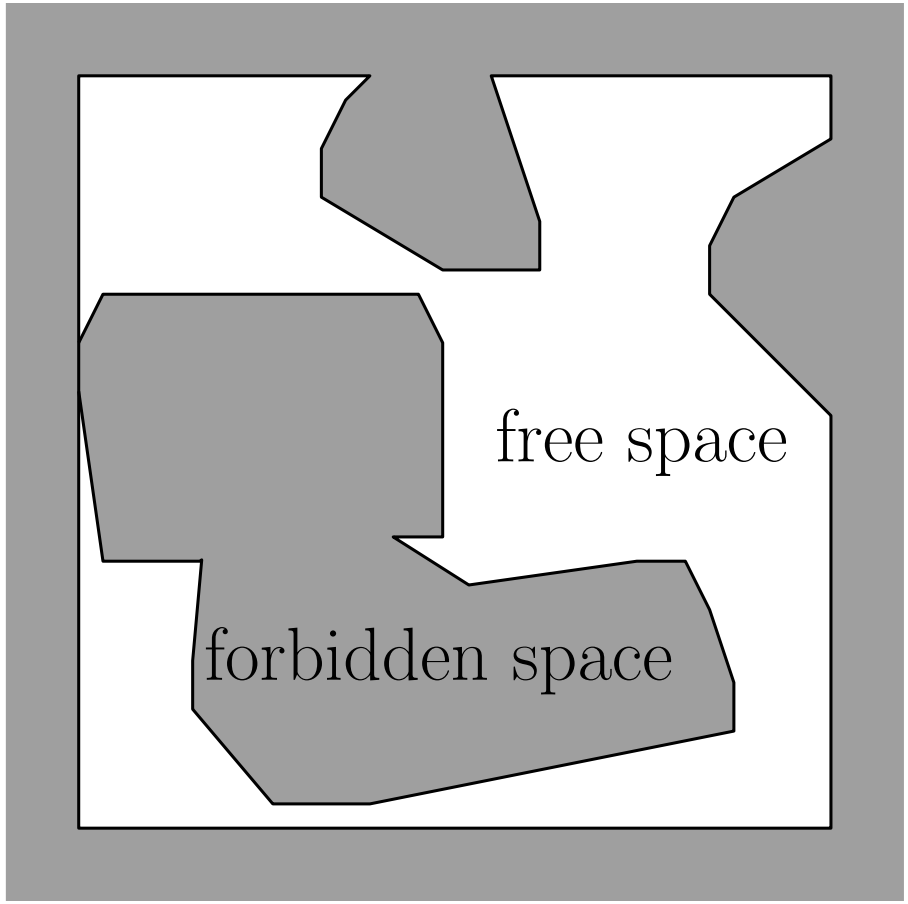




The configuration-space method

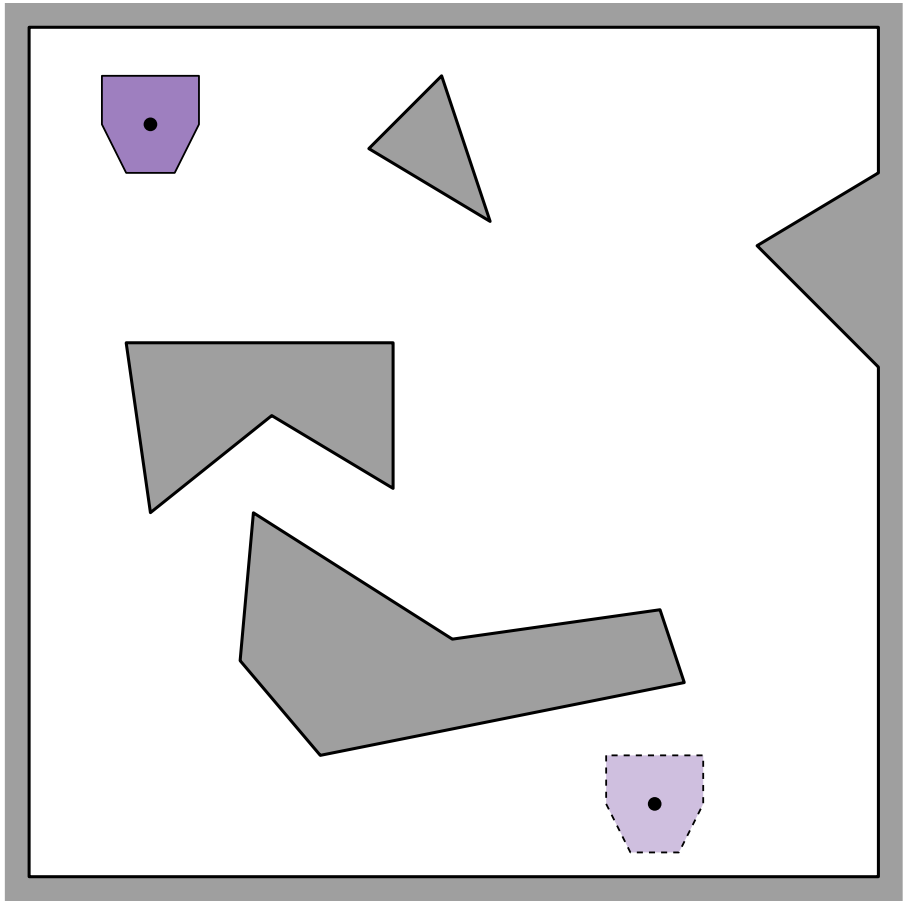


work space



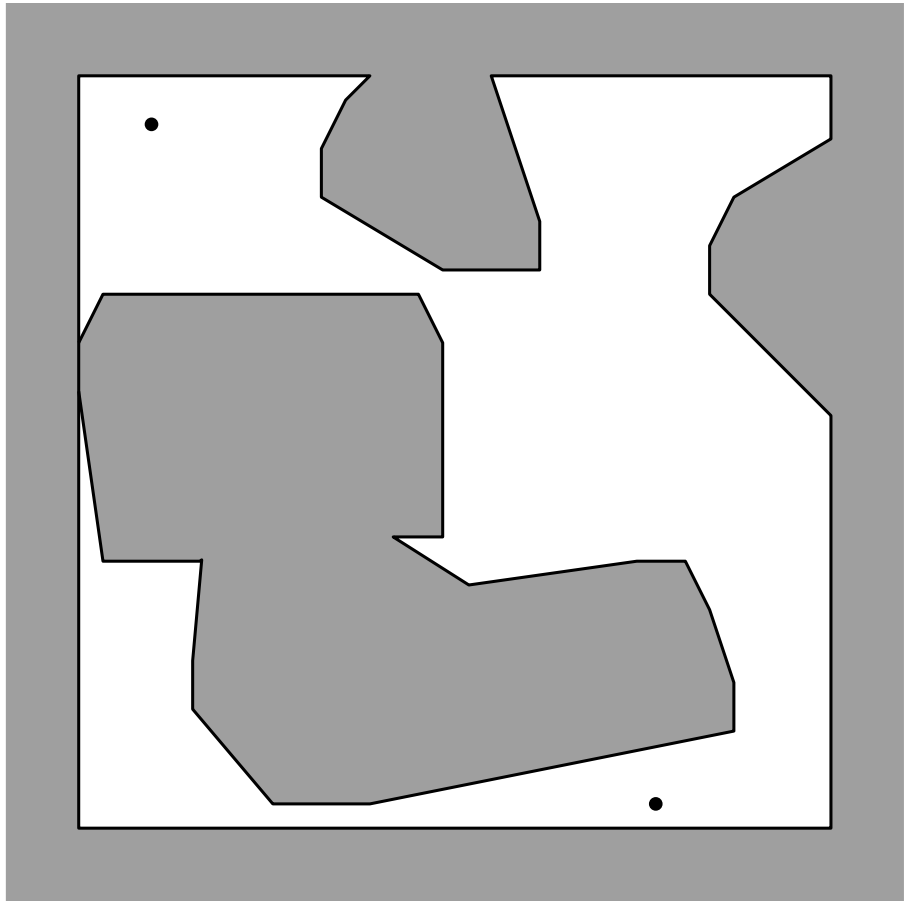
configuration space

The configuration-space method



work space

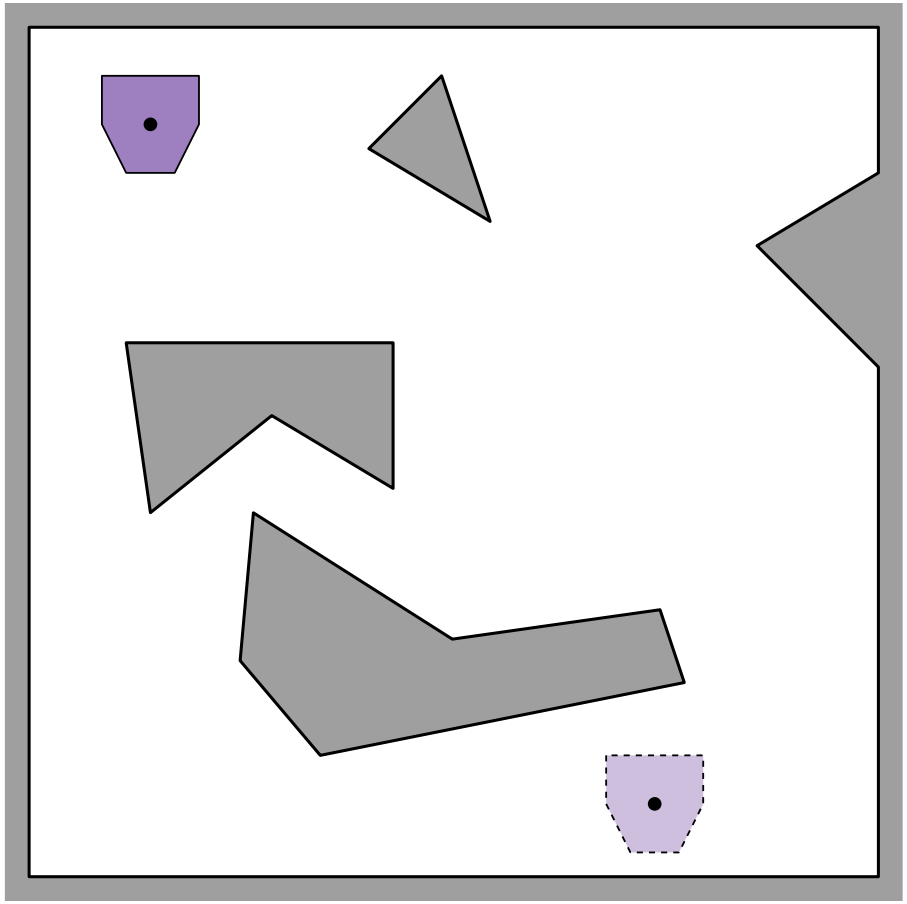
configuration = 2D position



configuration space

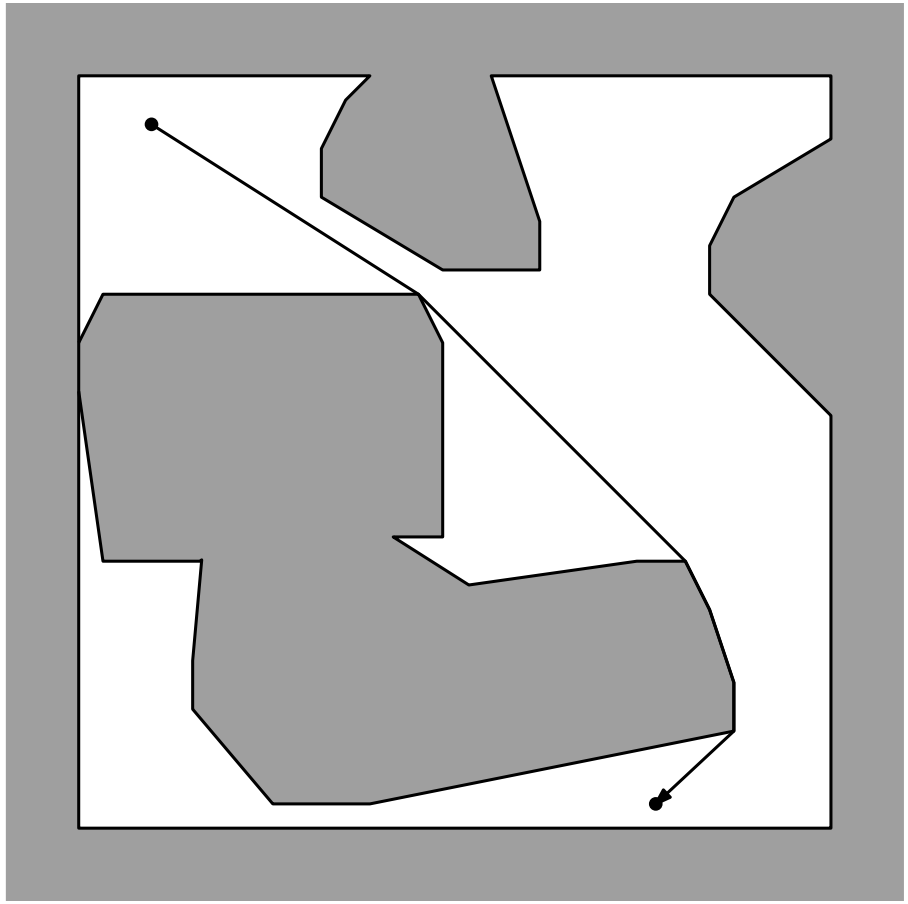
point in 2D

The configuration-space method



work space

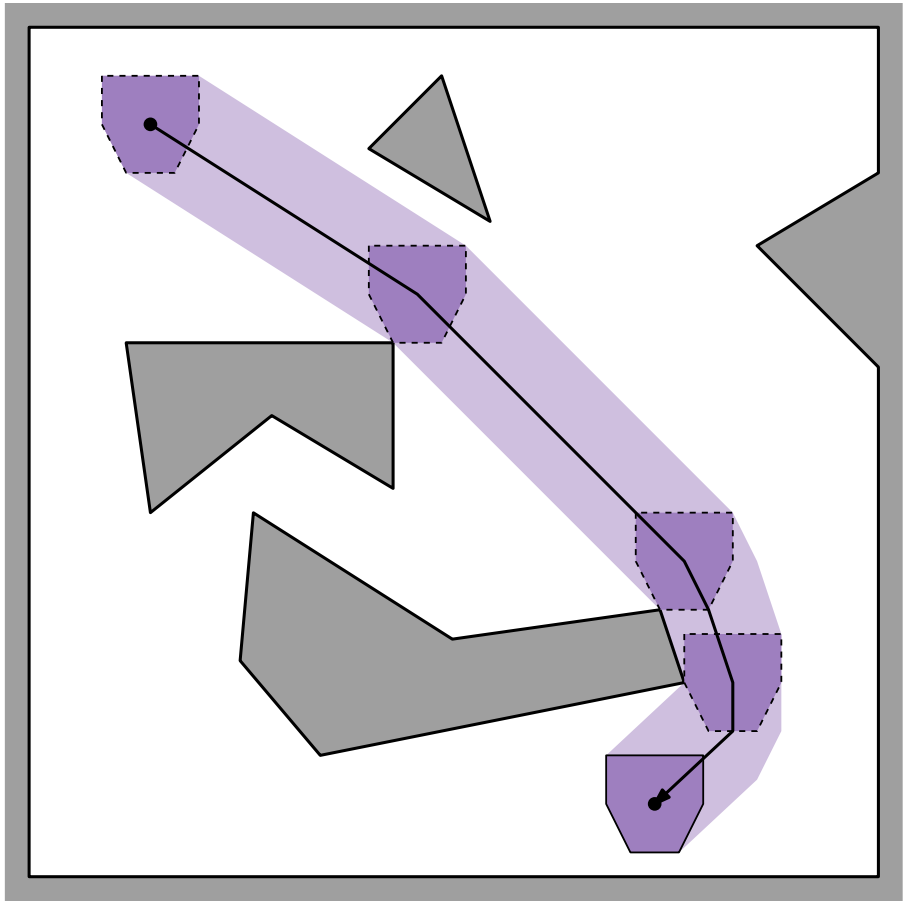
configuration = 2D position



configuration space

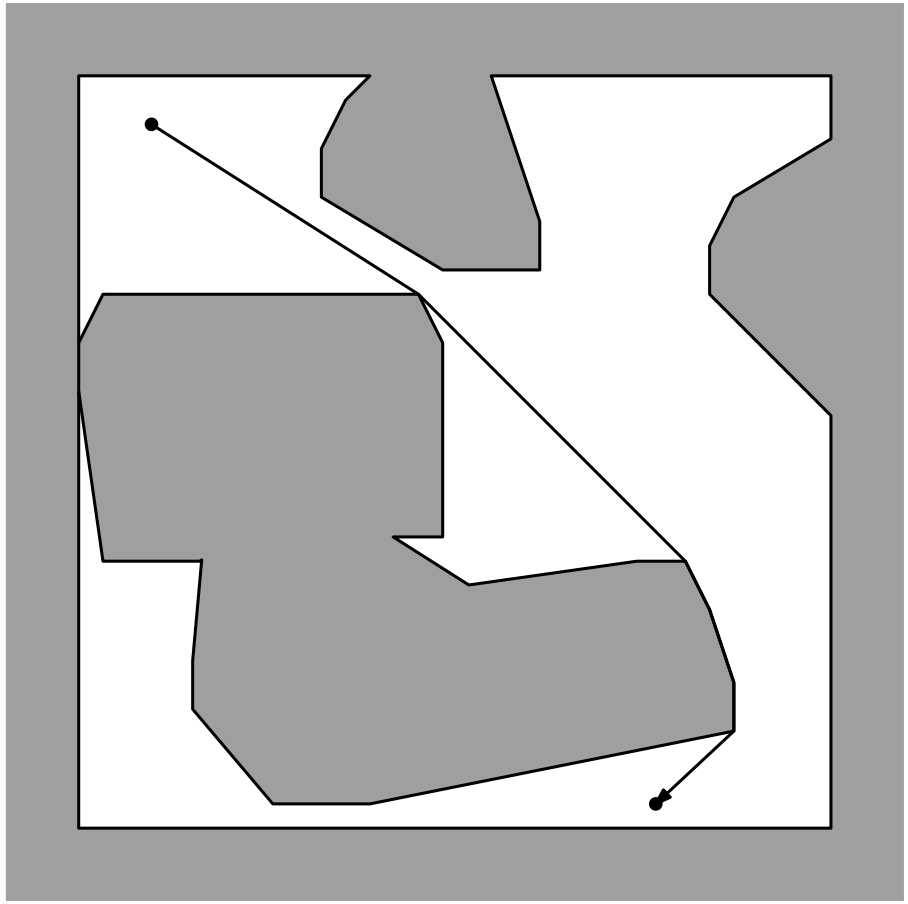
point in 2D

The configuration-space method



work space

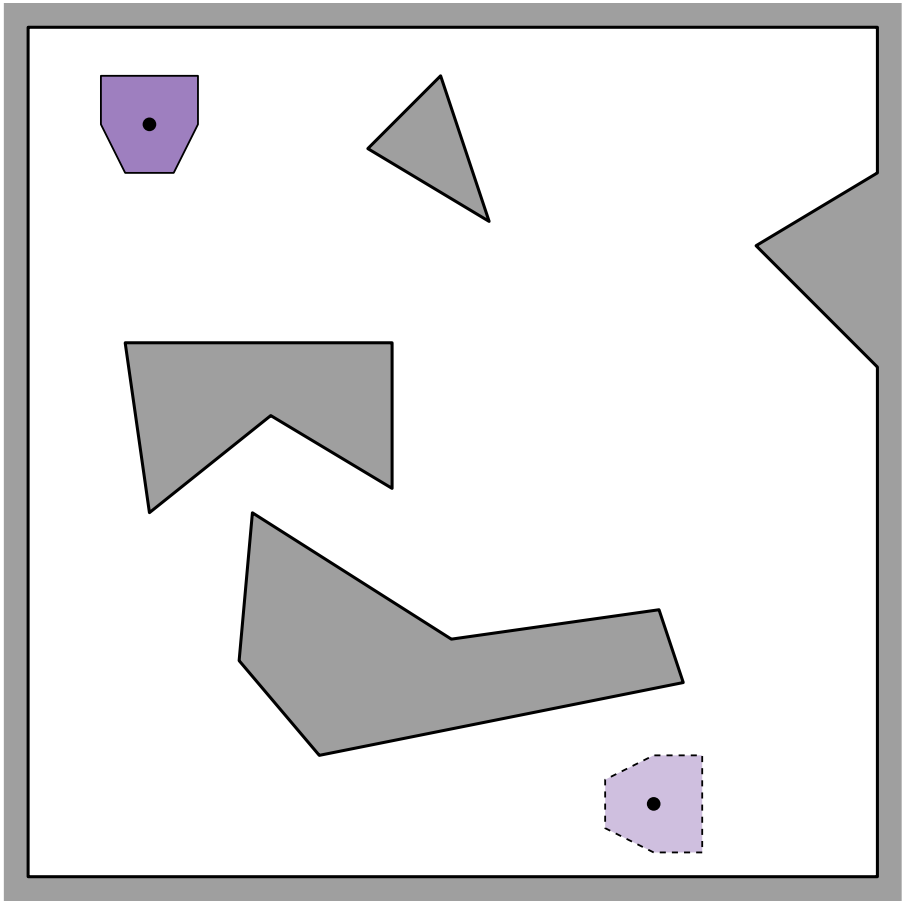
configuration = 2D position



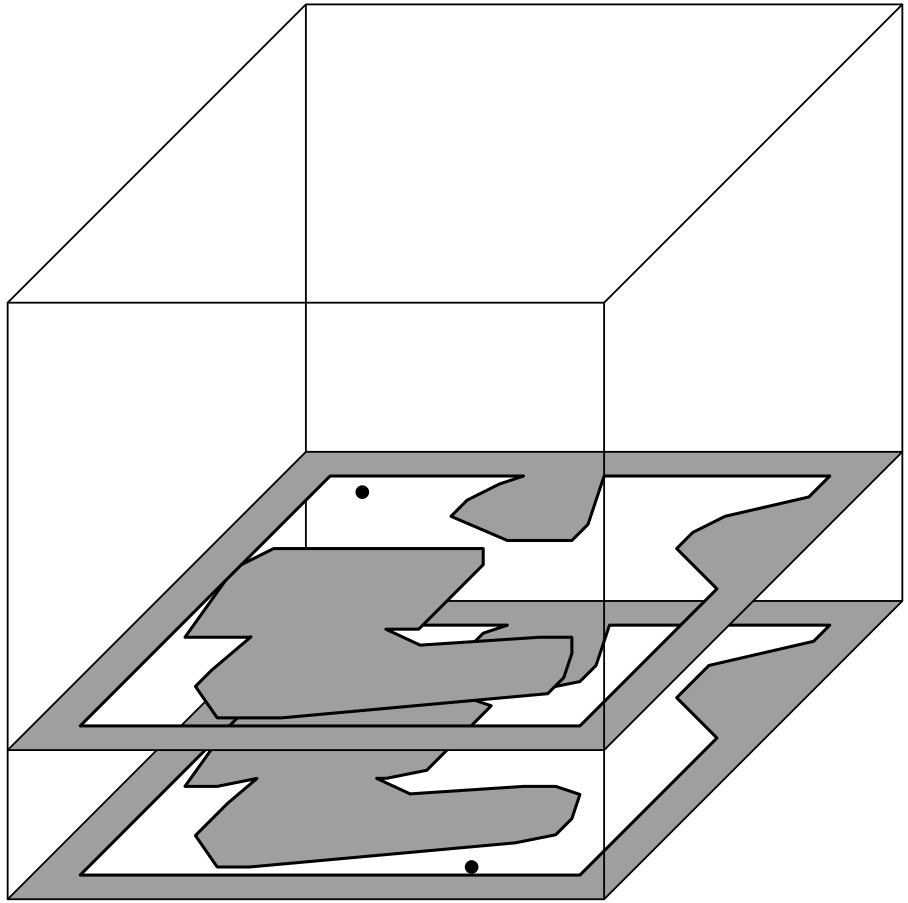
configuration space

point in 2D

The configuration-space method



work space

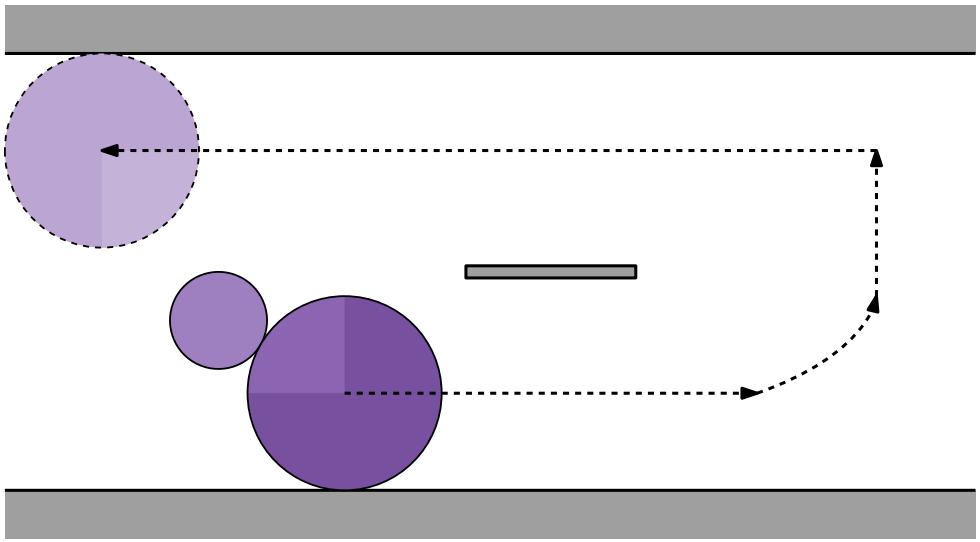


configuration space

configuration = 2D position +
1D orientation

point in 3D

The configuration-space method



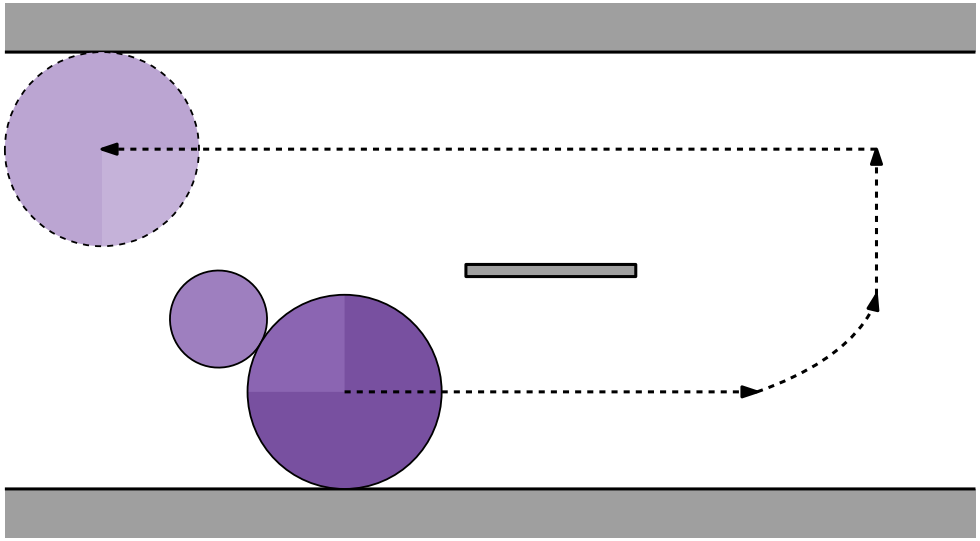
work space

configuration =
2D object position +
2D pusher position

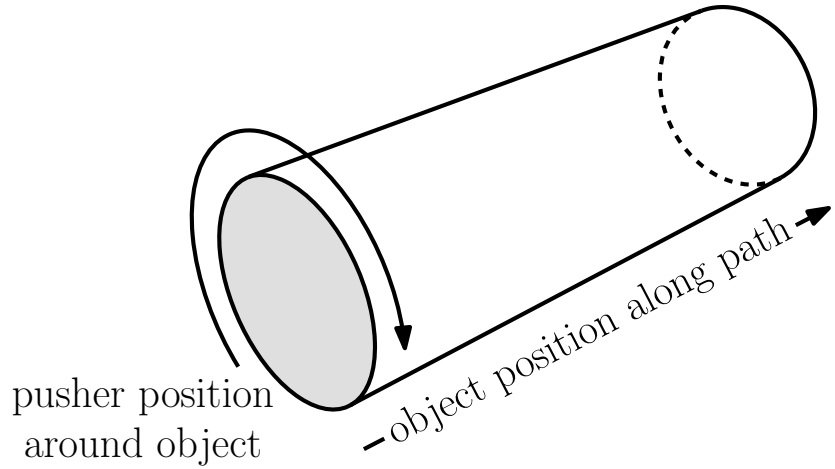
configuration space

point in 4D ?!

The configuration-space method



work space

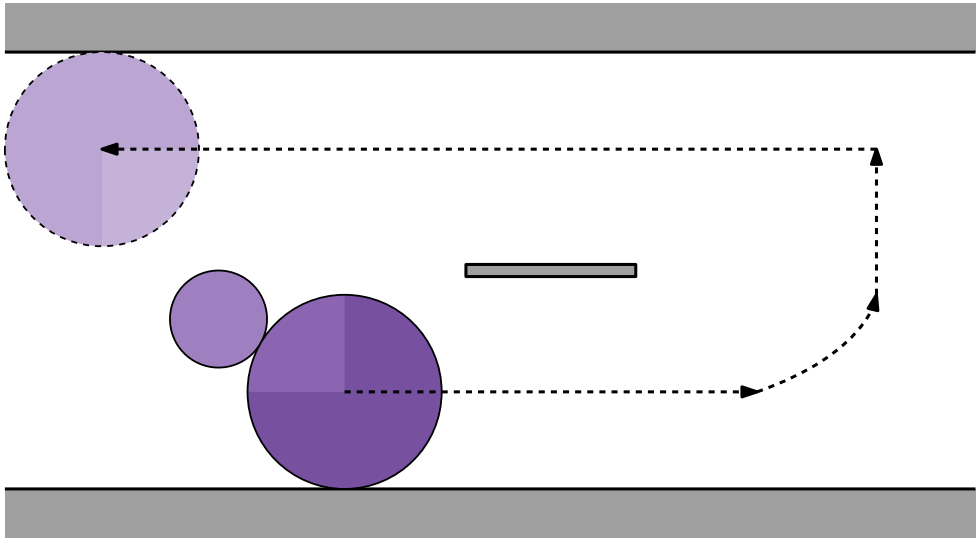


configuration space

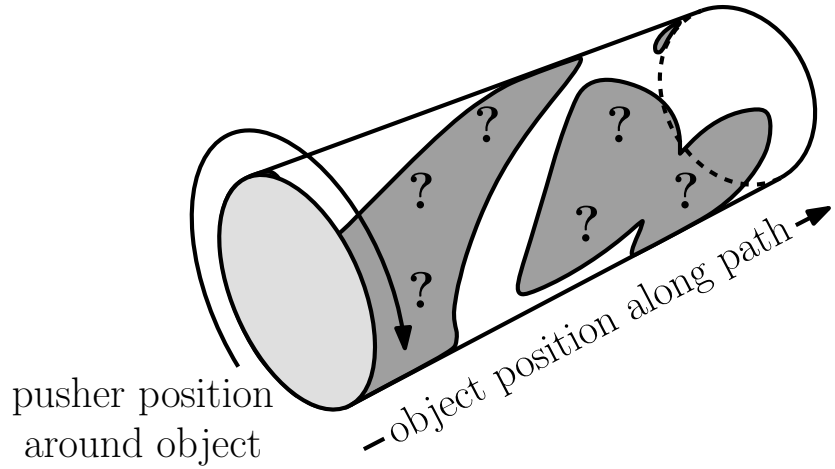
configuration =
1D object position along path +
1D pusher position around object

point in 2D (on cylinder)

The configuration-space method



work space

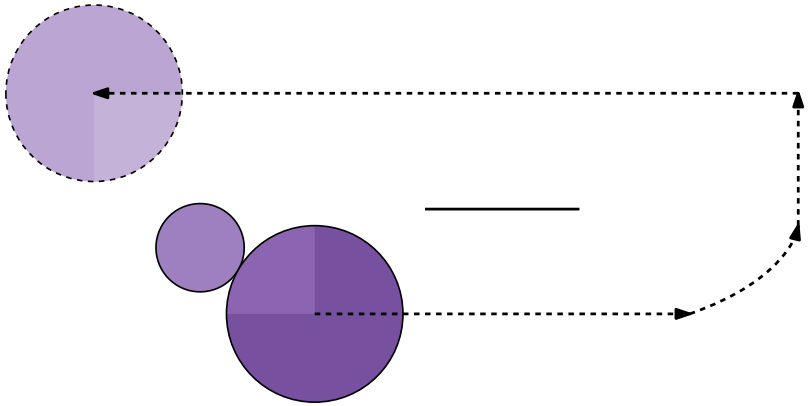


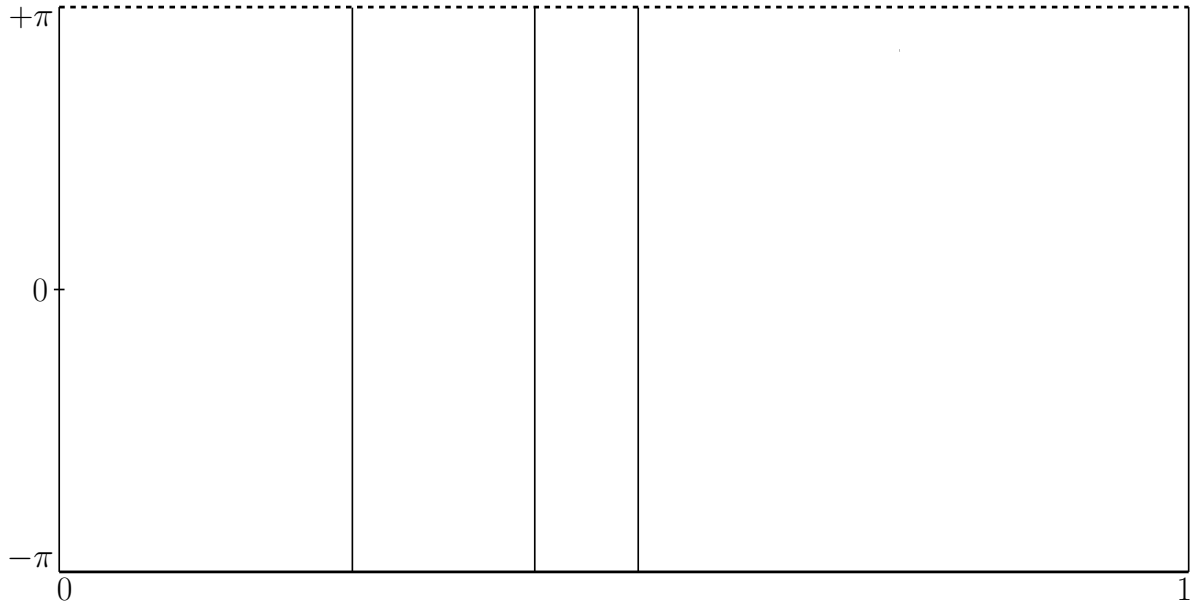
configuration space

configuration =
1D object position along path +
1D pusher position around object

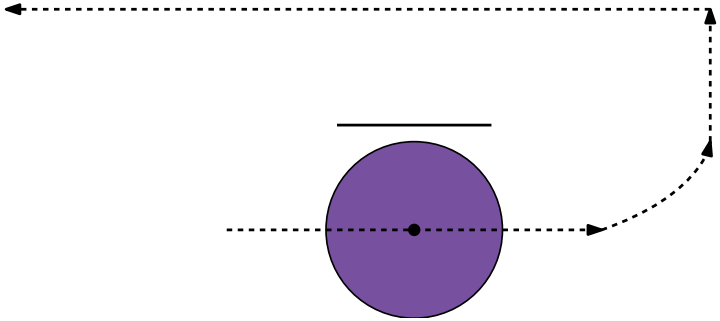
point in 2D (on cylinder)

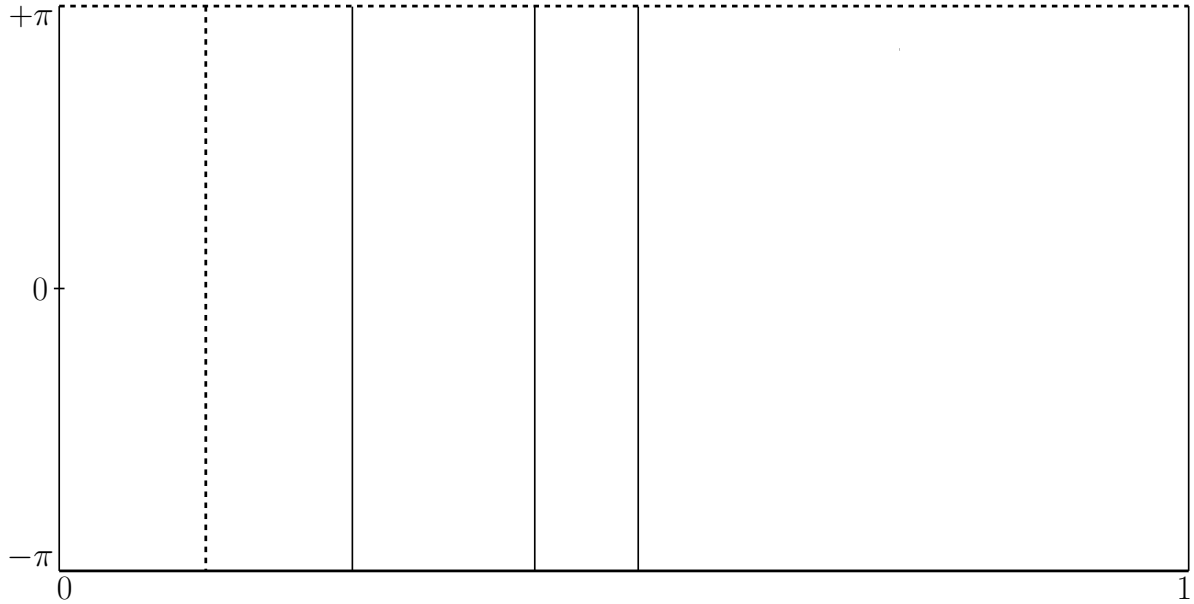
Configuration-space obstacles



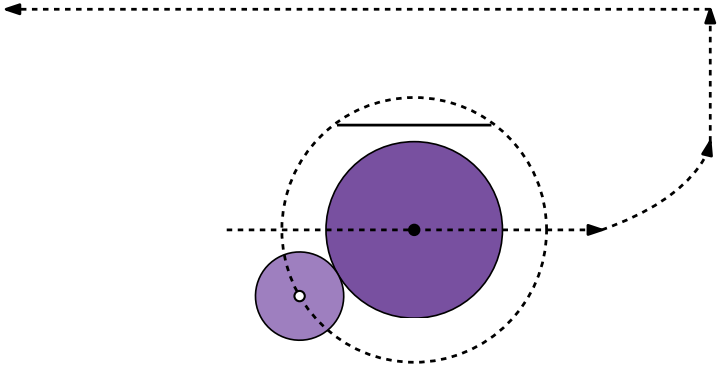


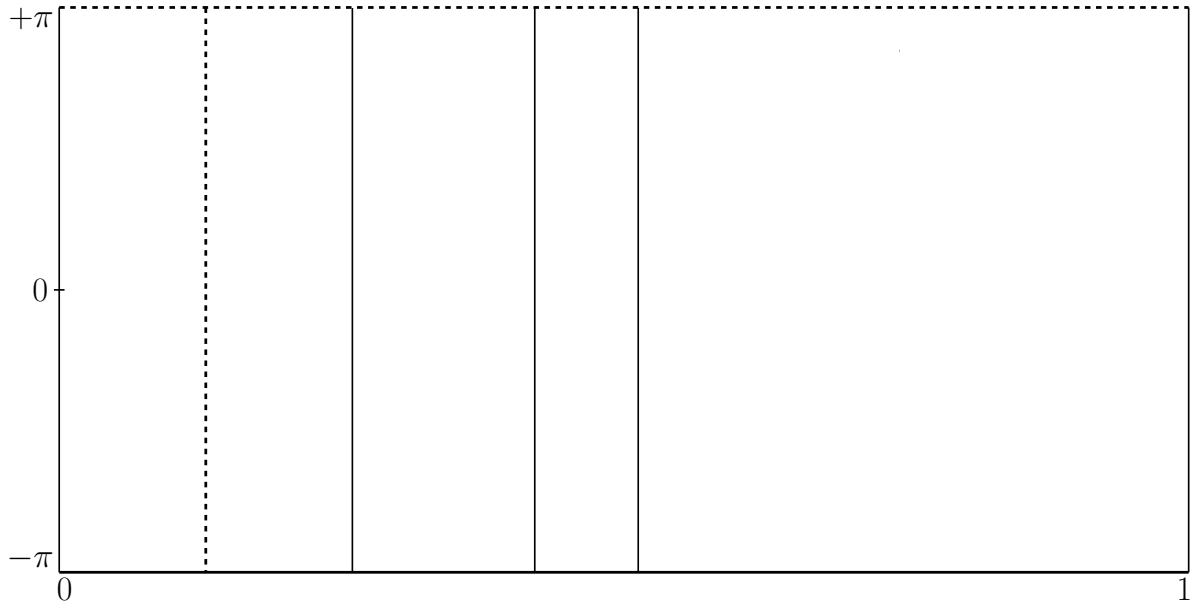
Configuration-space obstacles



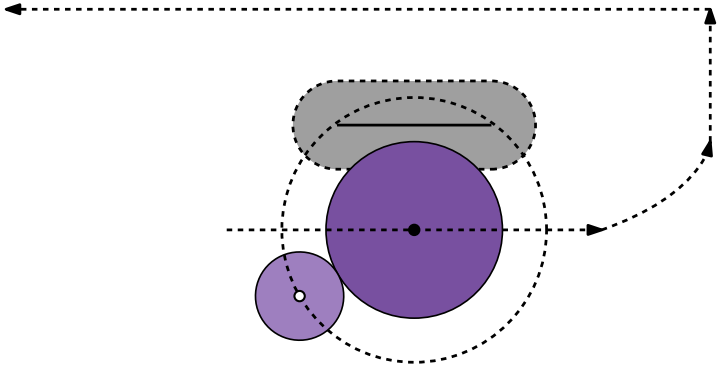


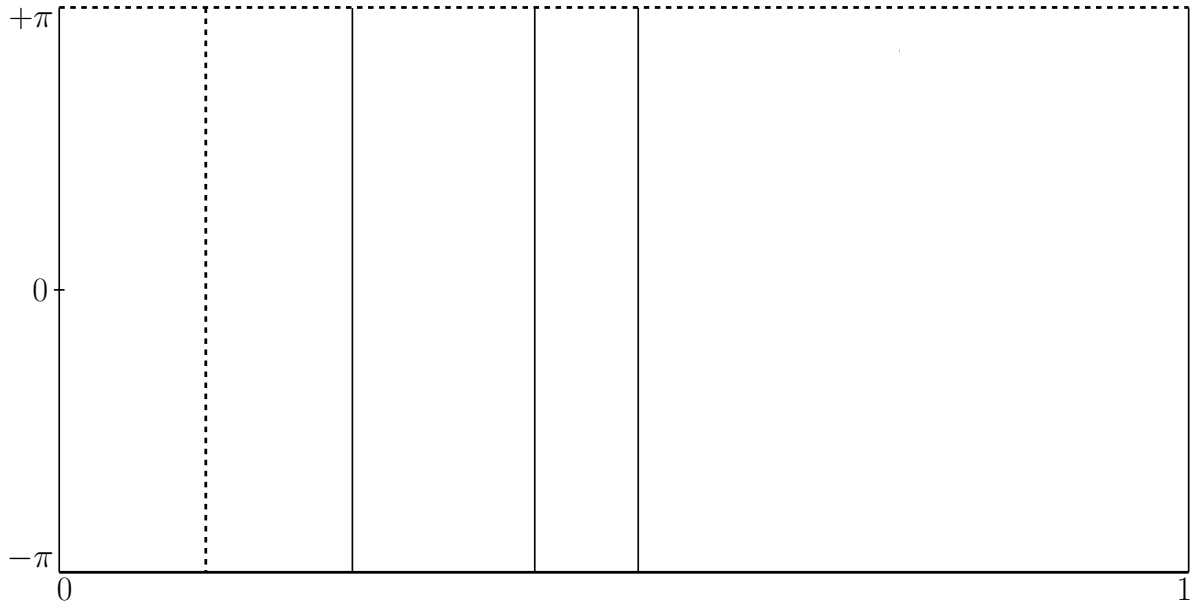
Configuration-space obstacles



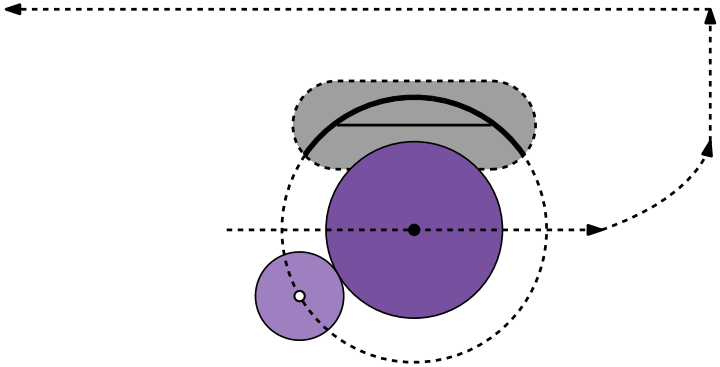


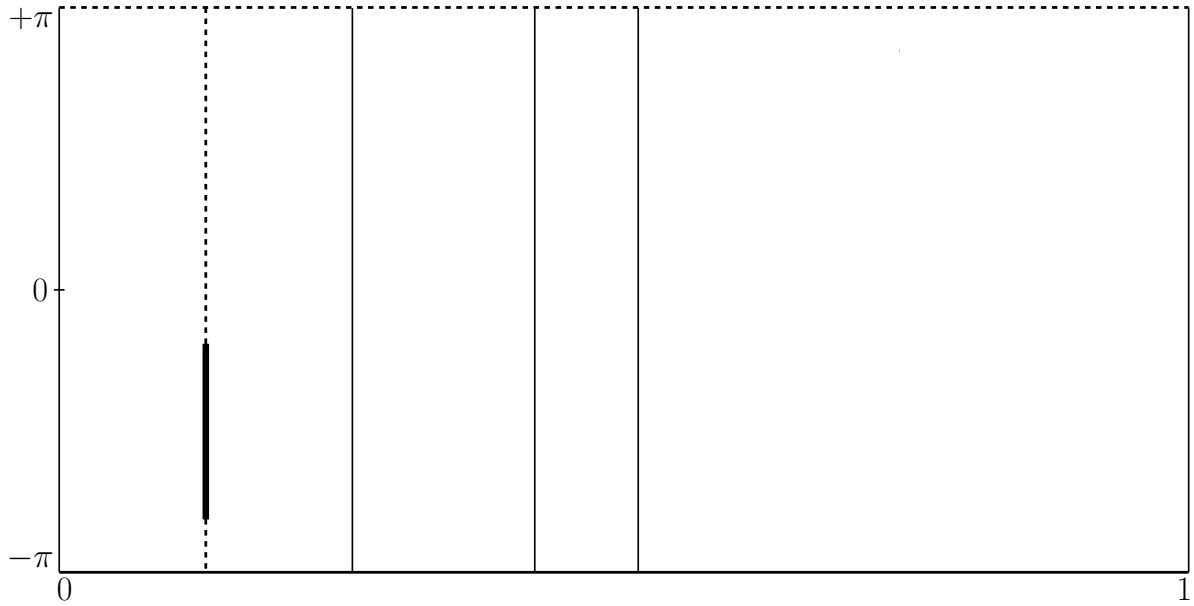
Configuration-space obstacles



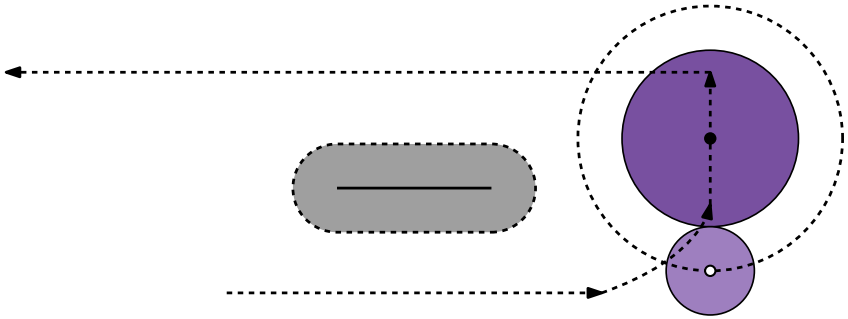


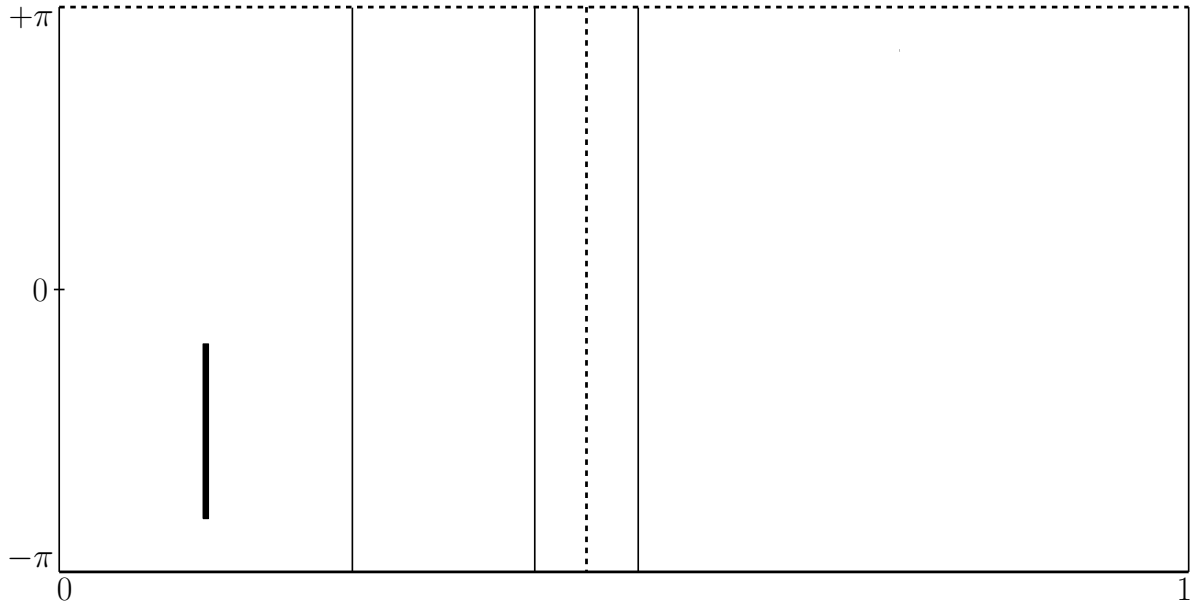
Configuration-space obstacles



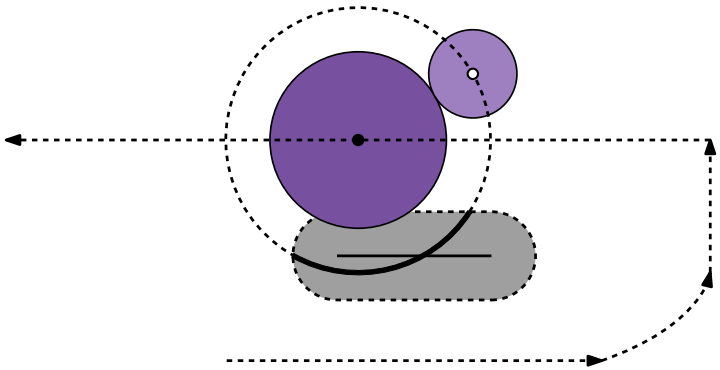


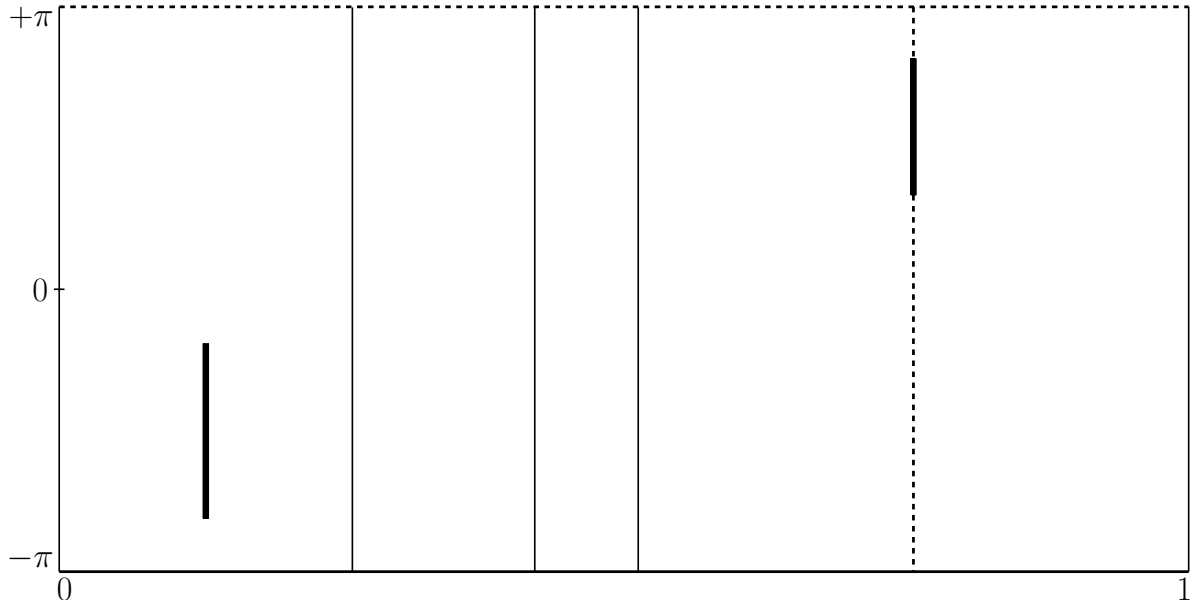
Configuration-space obstacles



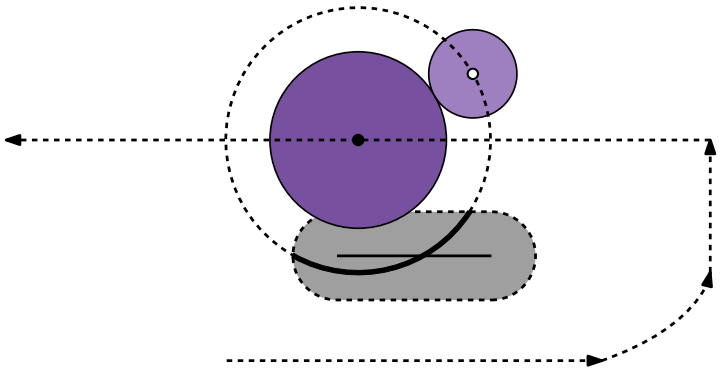


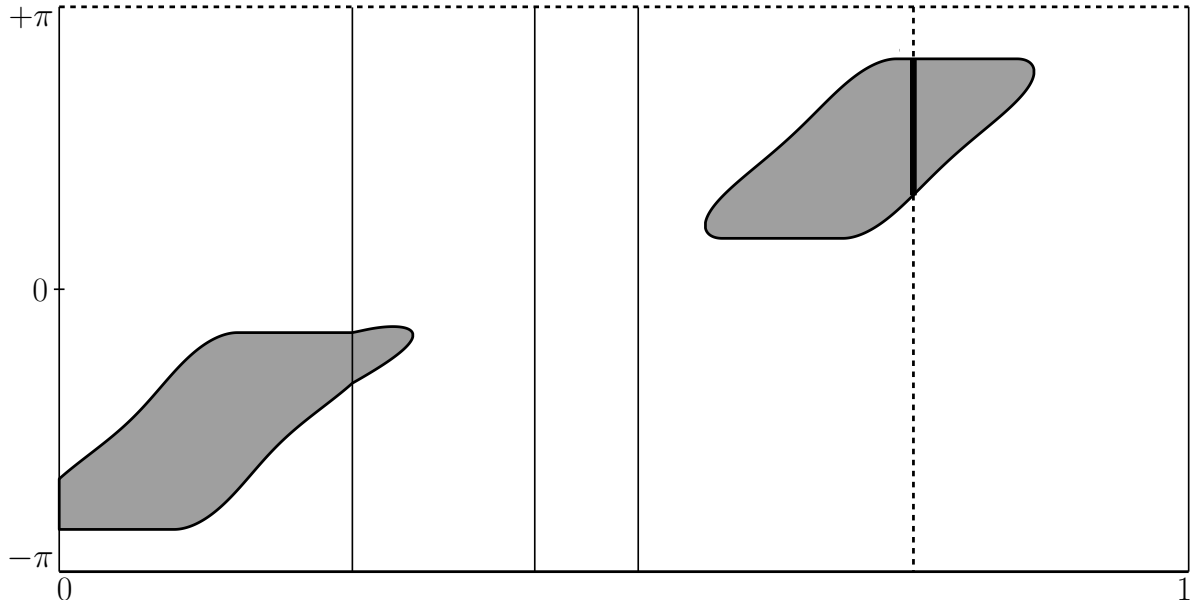
Configuration-space obstacles



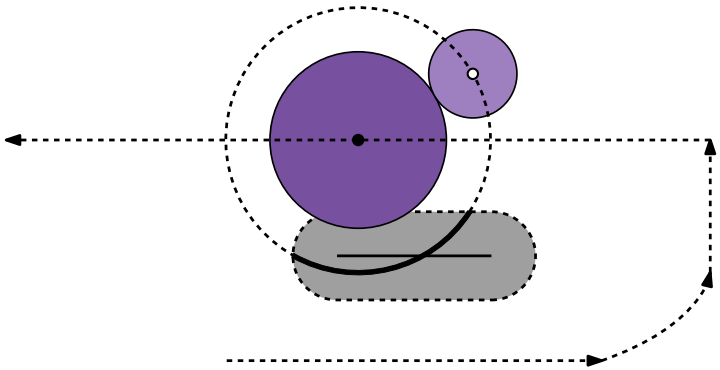


Configuration-space obstacles



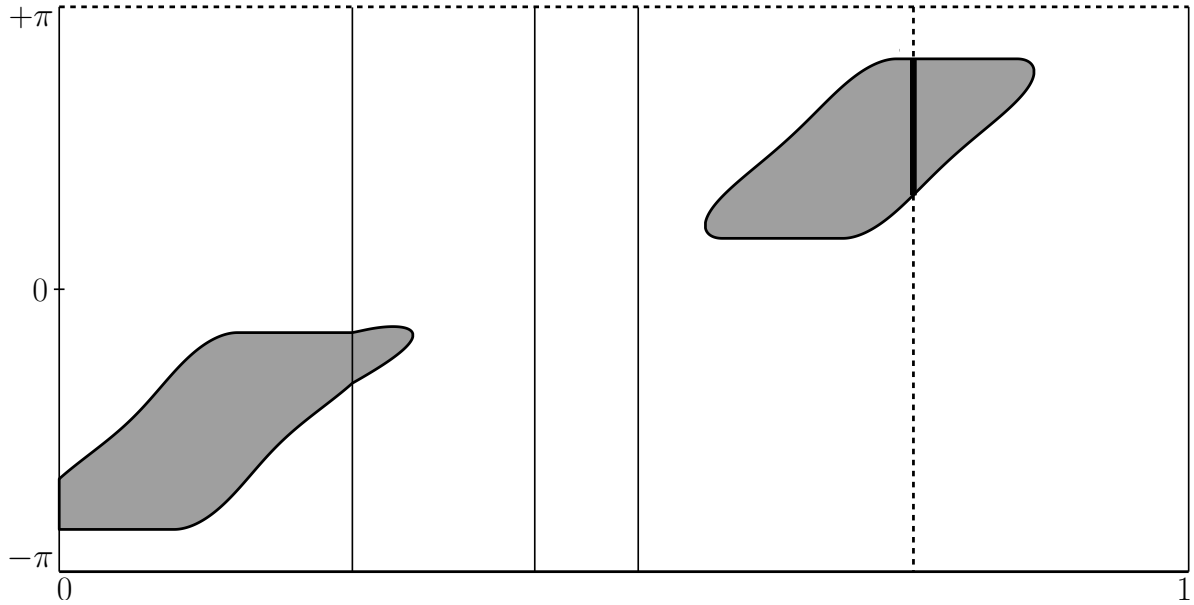


Configuration-space obstacles

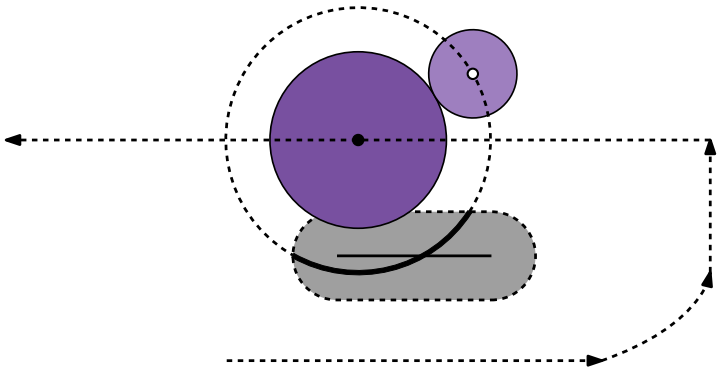


One $\mathcal{C}$-space obstacle:

- can have multiple components

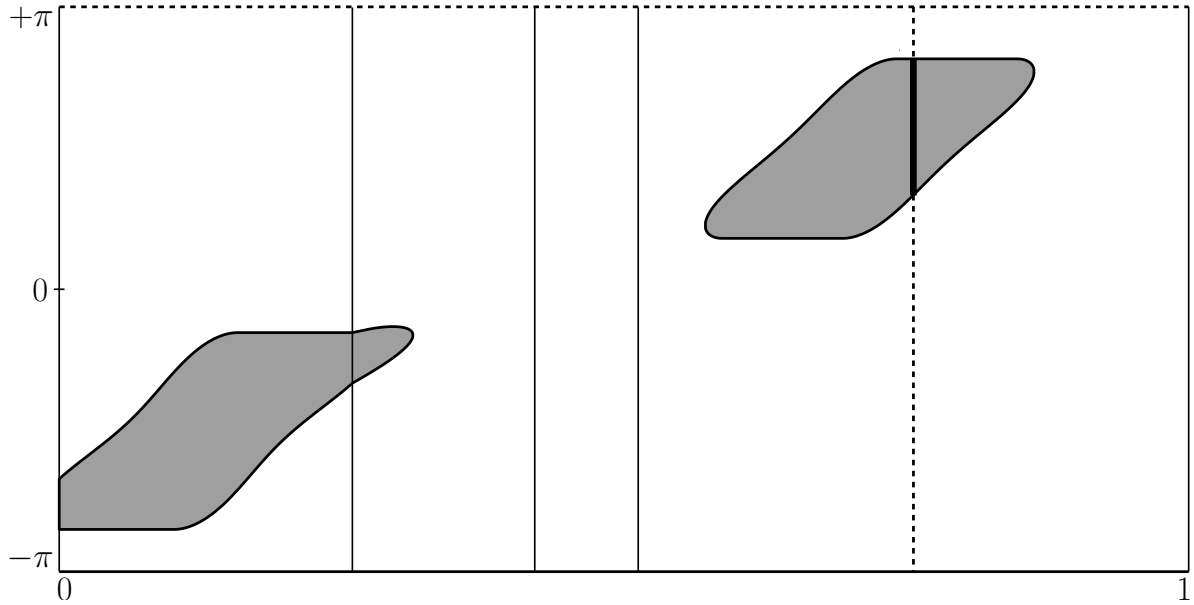


Configuration-space obstacles

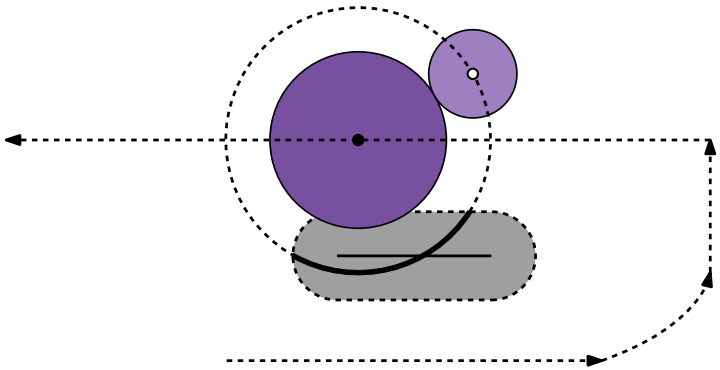


One $\mathcal{C}$-space obstacle:

- can have multiple components, but total complexity $O(\#\text{sections})$

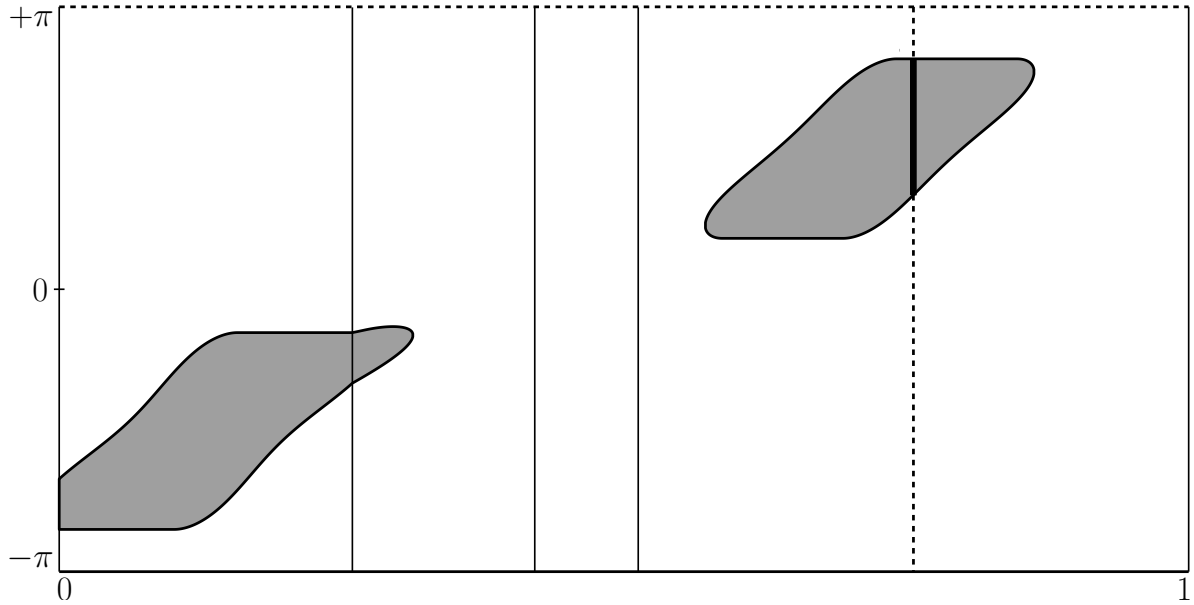


Configuration-space obstacles

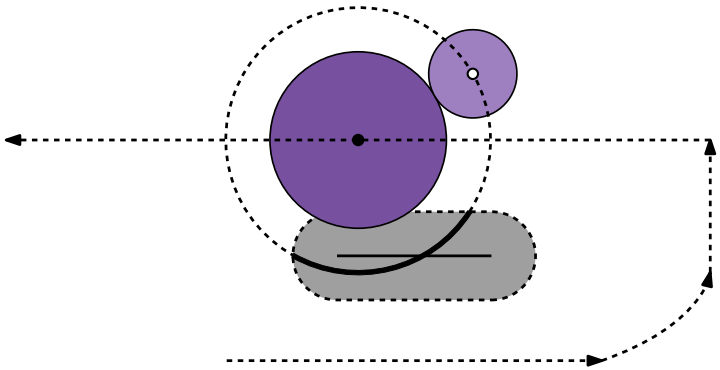


One $\mathcal{C}$-space obstacle:

- can have multiple components, but total complexity $O(\#\text{sections})$
- "weird" curves

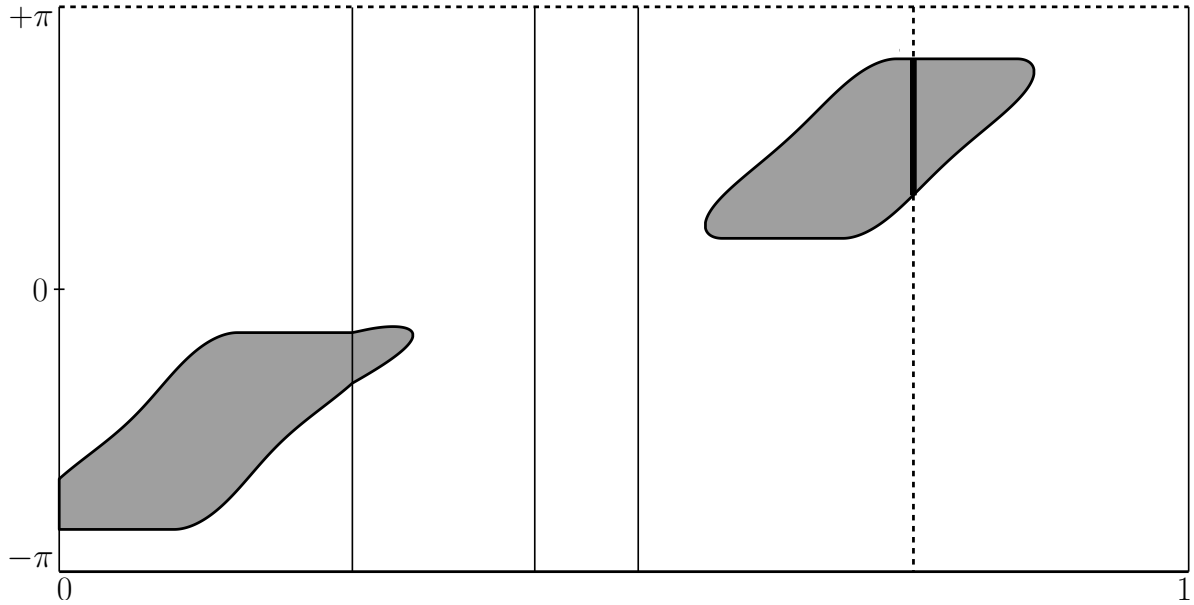


Configuration-space obstacles

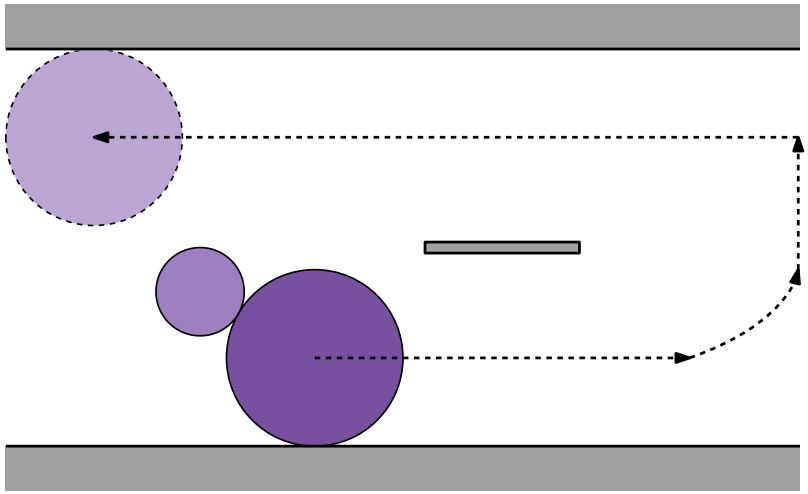


One $\mathcal{C}$-space obstacle:

- can have multiple components, but total complexity $O(\#\text{sections})$
- "weird" curves, but can be handled exactly with smart representation



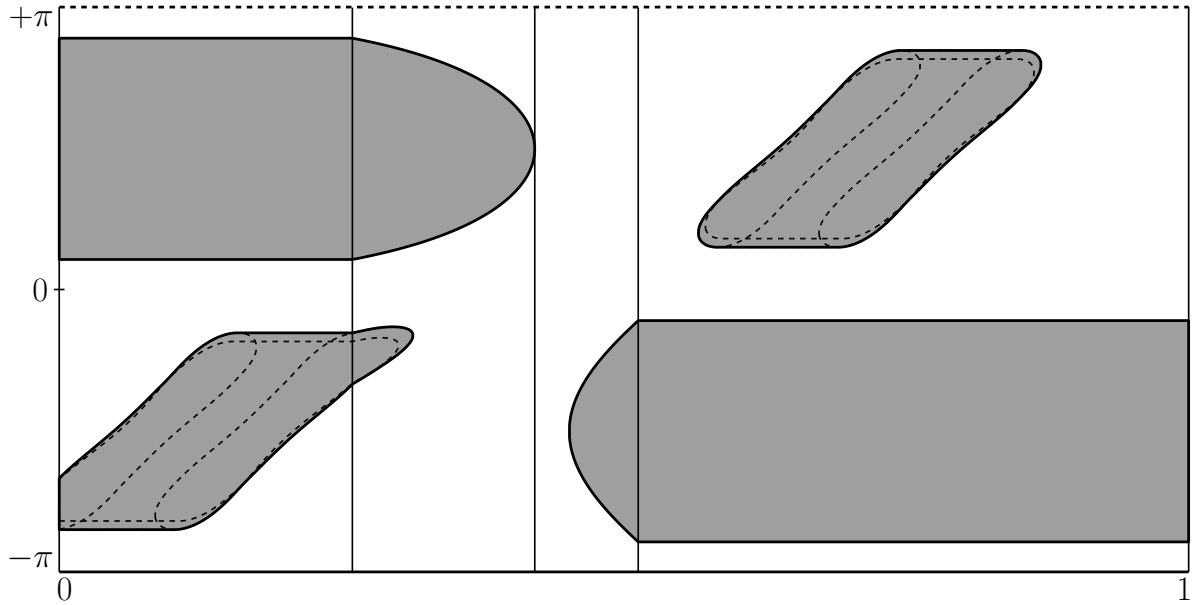
Configuration-space obstacles



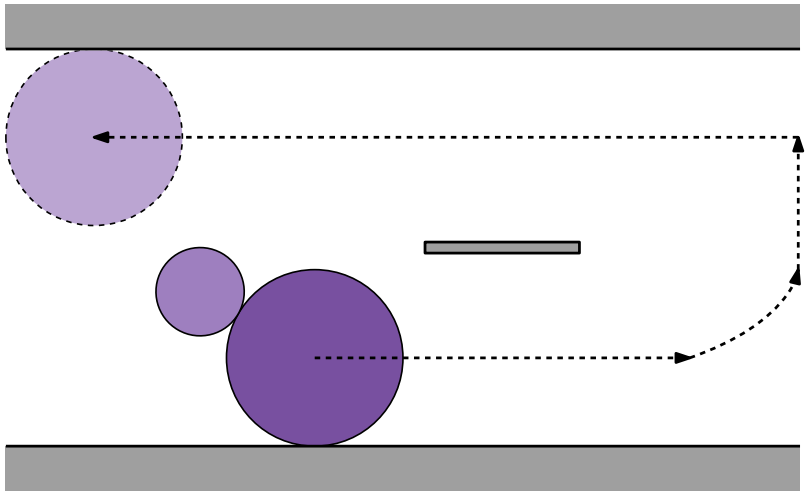
One $\mathcal{C}$-space obstacle:

- can have multiple components, but total complexity $O(\#\text{sections})$
- "weird" curves, but can be handled exactly with smart representation

Union of all $\mathcal{C}$-space obstacles



Configuration-space obstacles

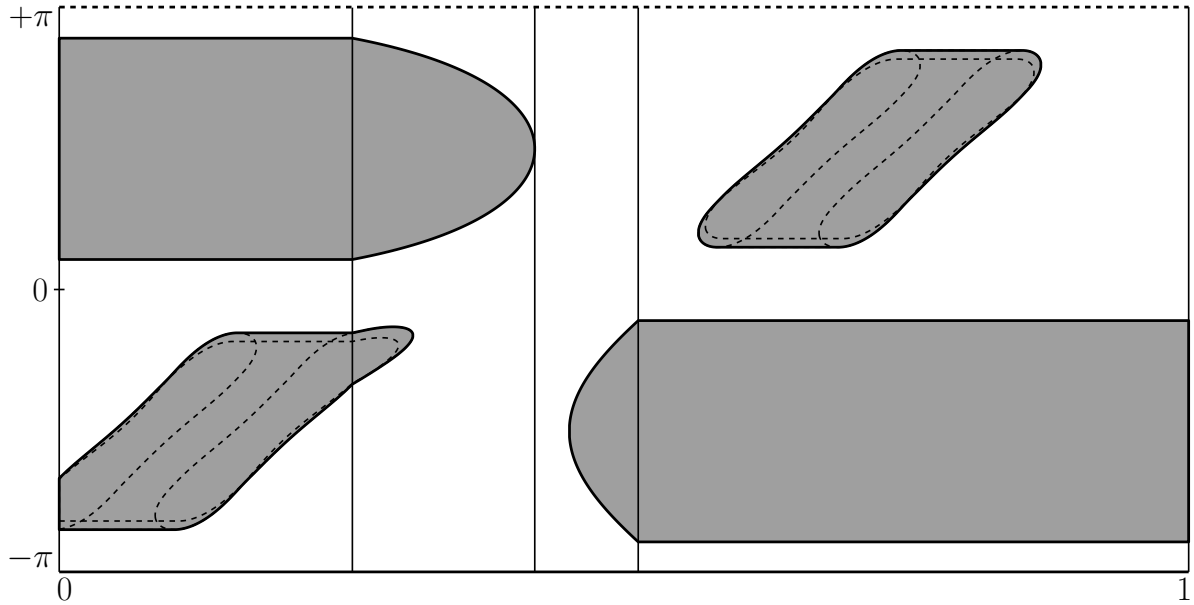


One $\mathcal{C}$-space obstacle:

- can have multiple components, but total complexity $O(\#\text{sections})$
- "weird" curves, but can be handled exactly with smart representation

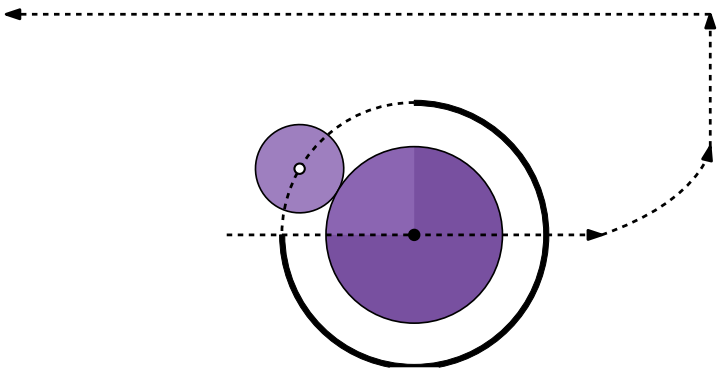
Union of all $\mathcal{C}$-space obstacles:

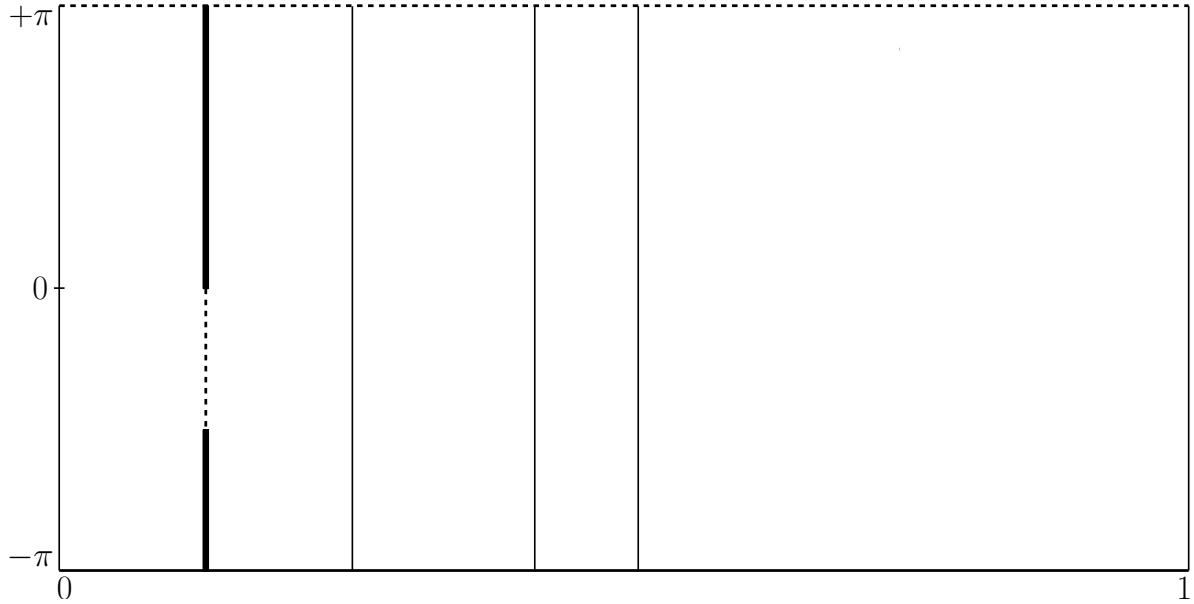
- linear complexity: $O(\#\text{obstacles} \times \#\text{sections})$



The forbidden push range

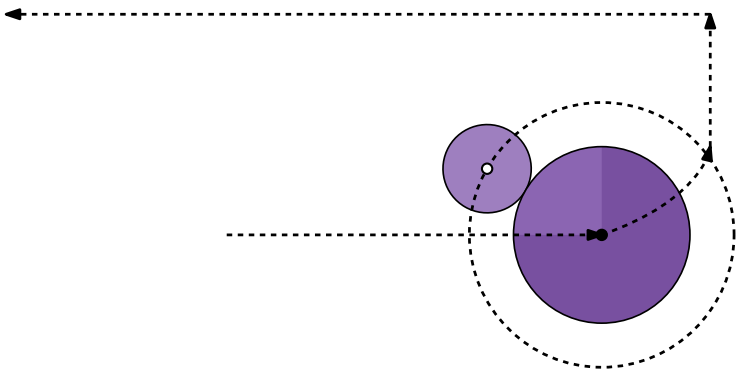
A configuration is also in the forbidden space if the pusher is outside the push range.

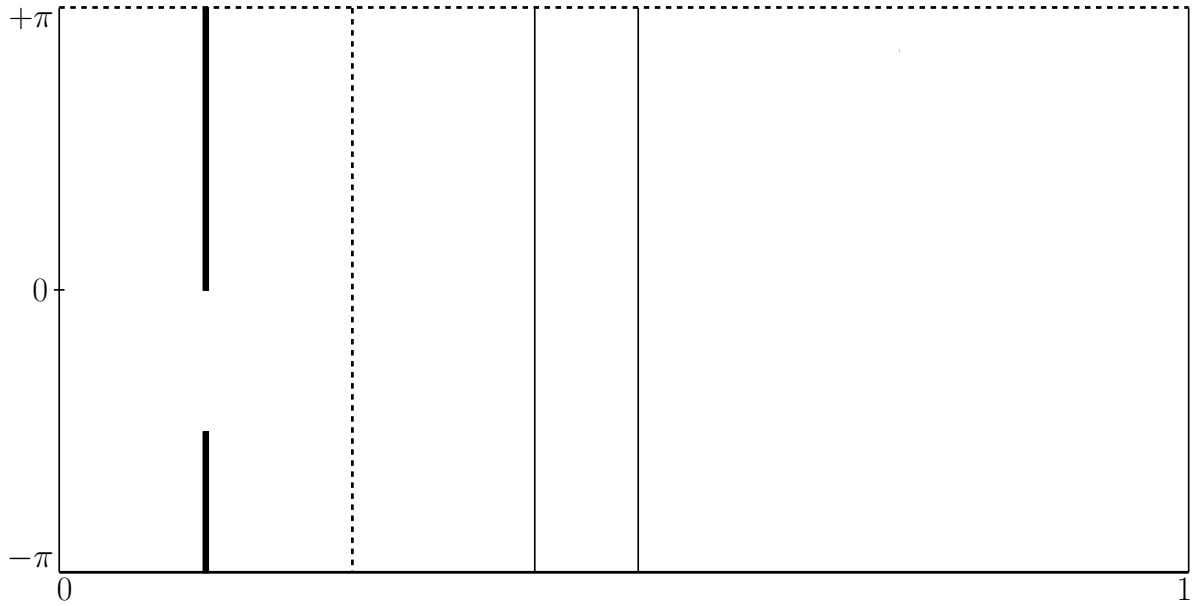




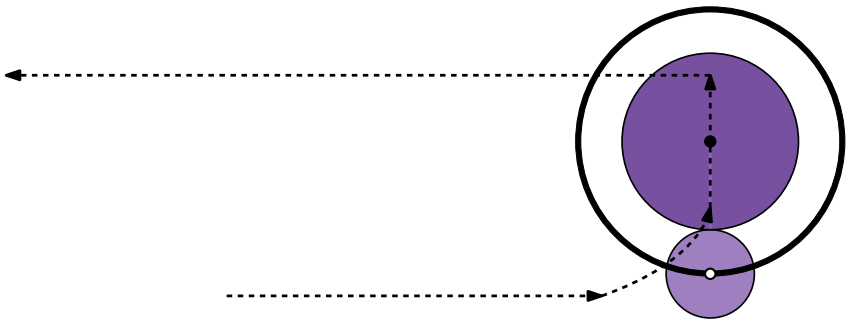
The forbidden push range

A configuration is also in the forbidden space if the pusher is outside the push range.

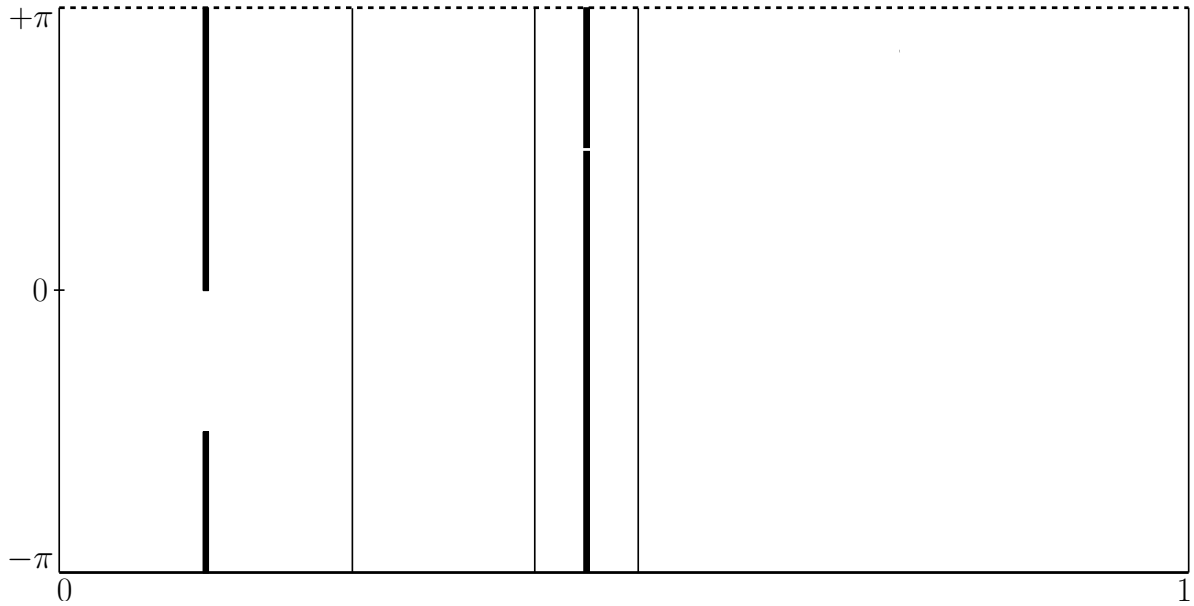




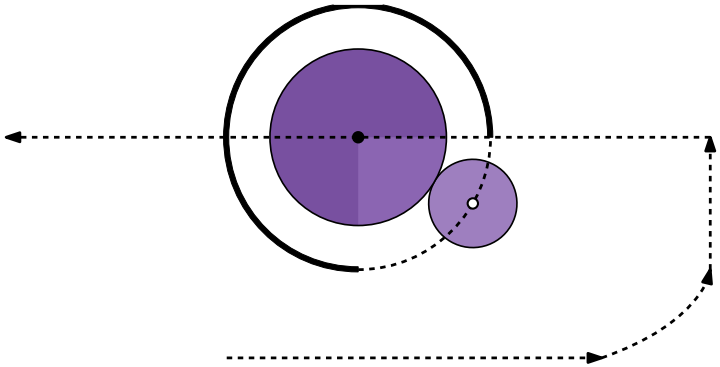
The forbidden push range



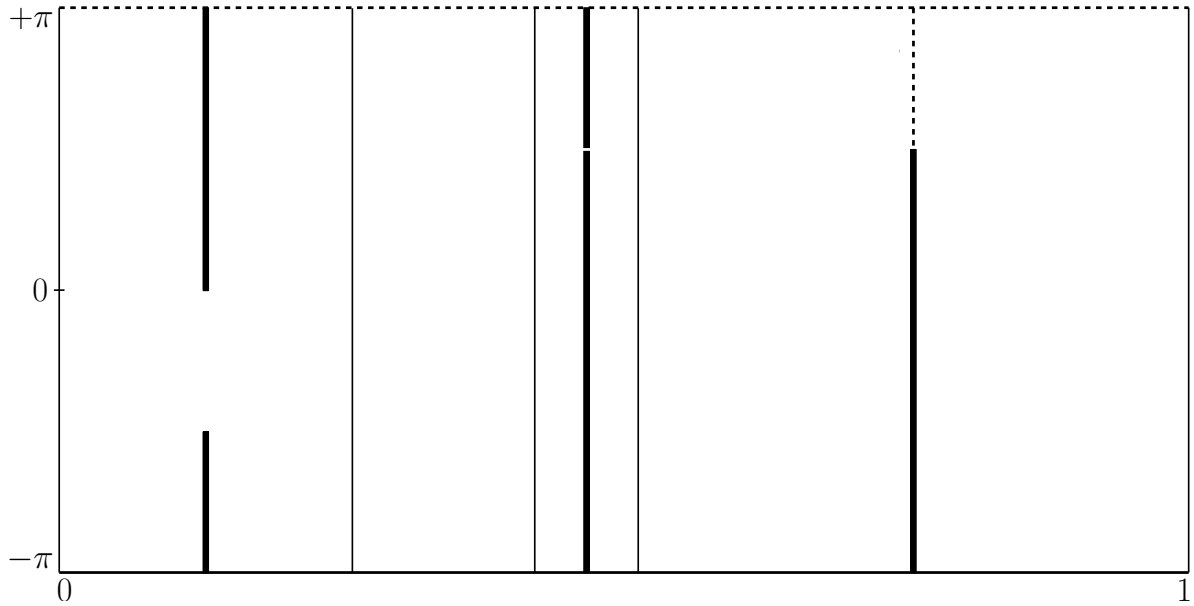
A configuration is also in the forbidden space if the pusher is outside the push range.



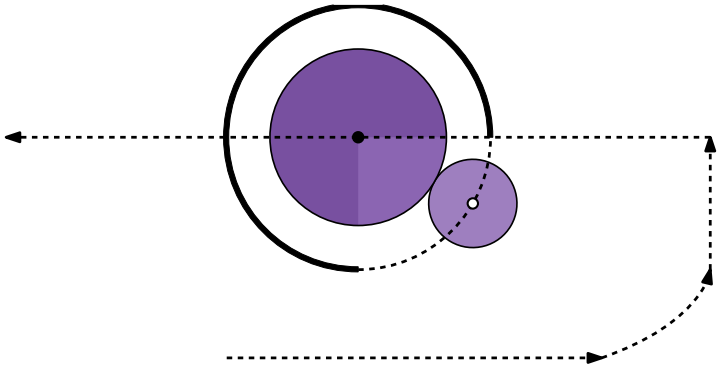
The forbidden push range



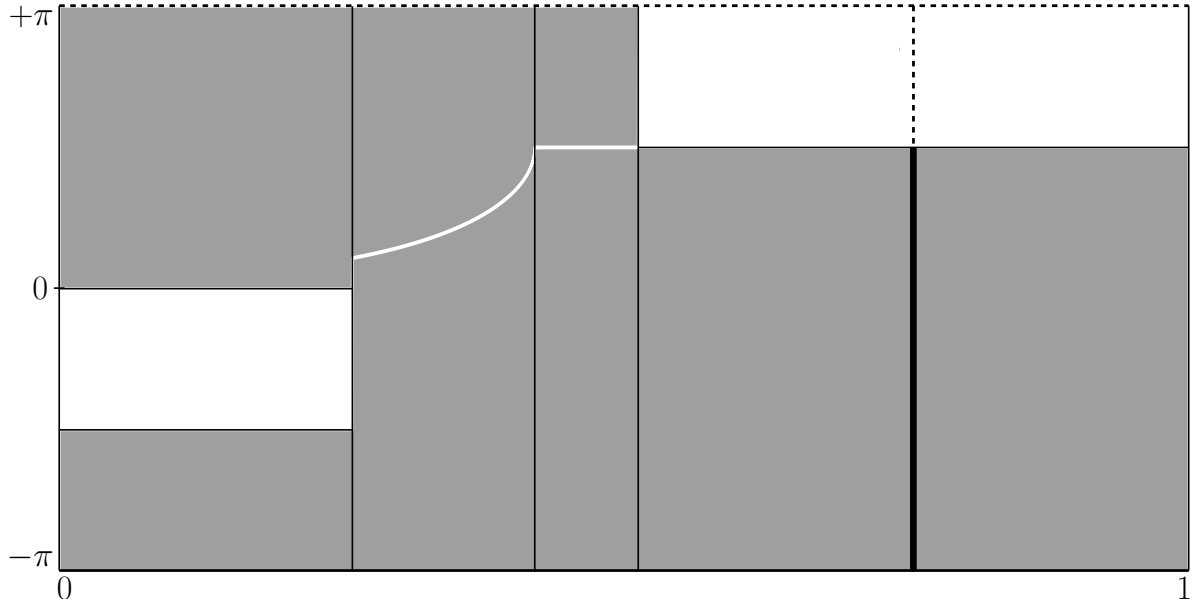
A configuration is also in the forbidden space if the pusher is outside the push range.



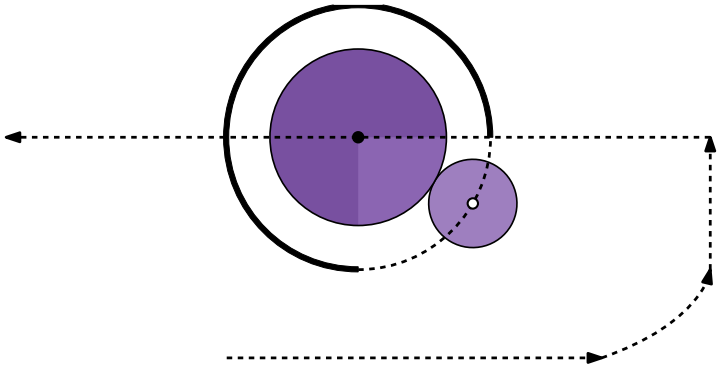
The forbidden push range



A configuration is also in the forbidden space if the pusher is outside the push range.

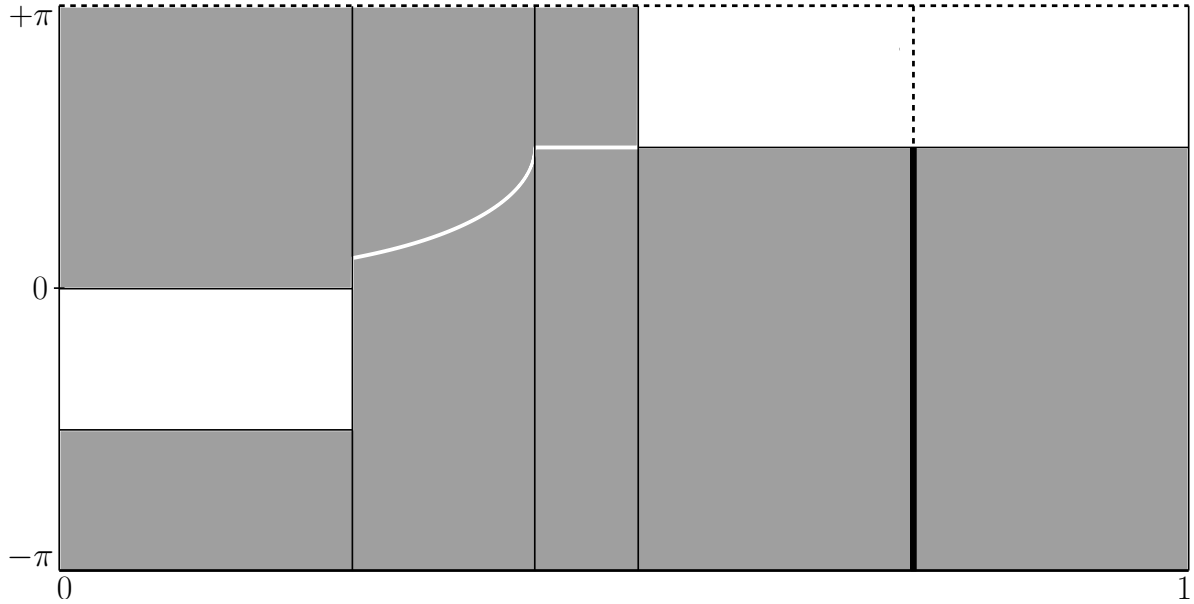


The forbidden push range

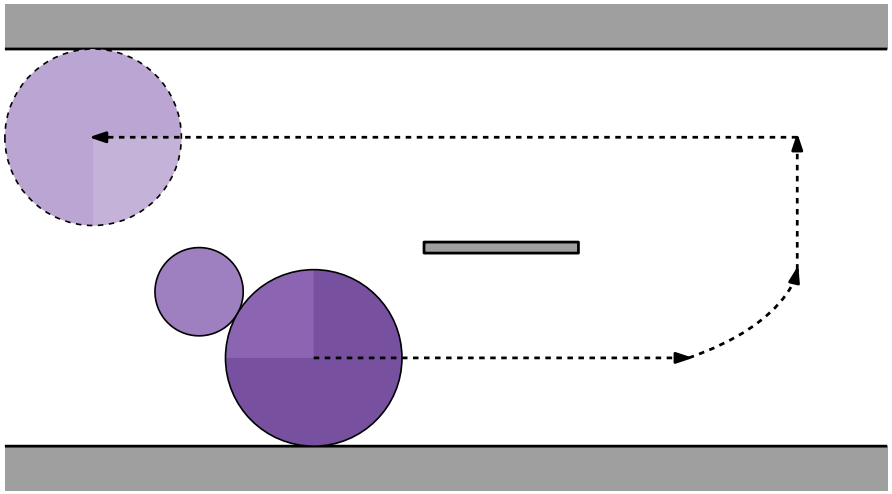


A configuration is also in the forbidden space if the pusher is outside the push range.

Forbidden push range also has $O(\#\text{sections})$ complexity.



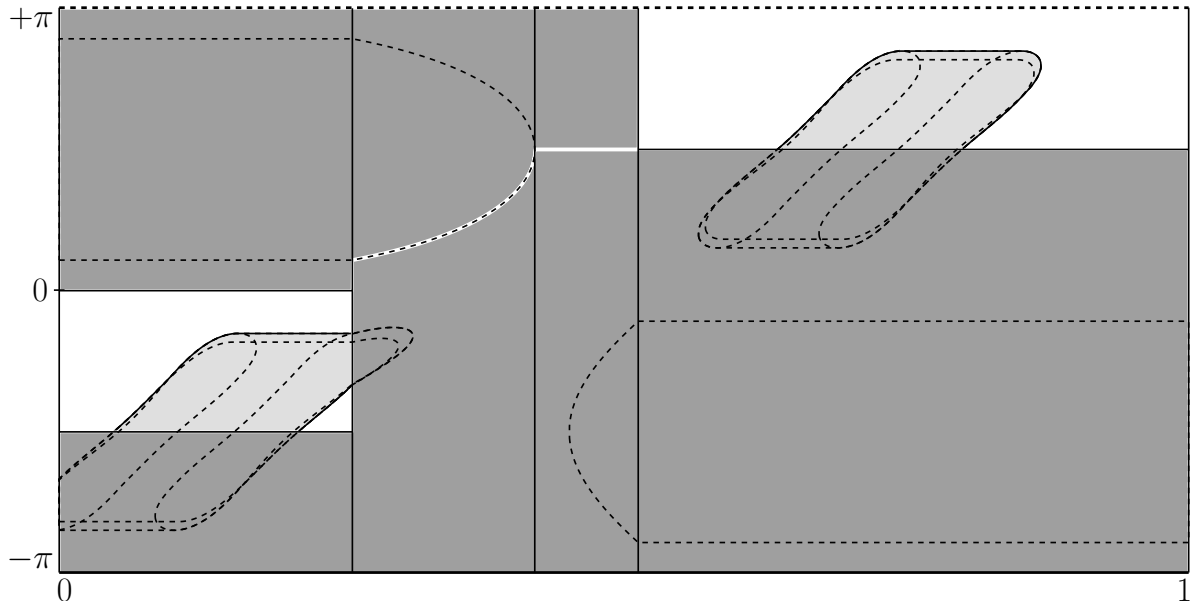
The forbidden push range



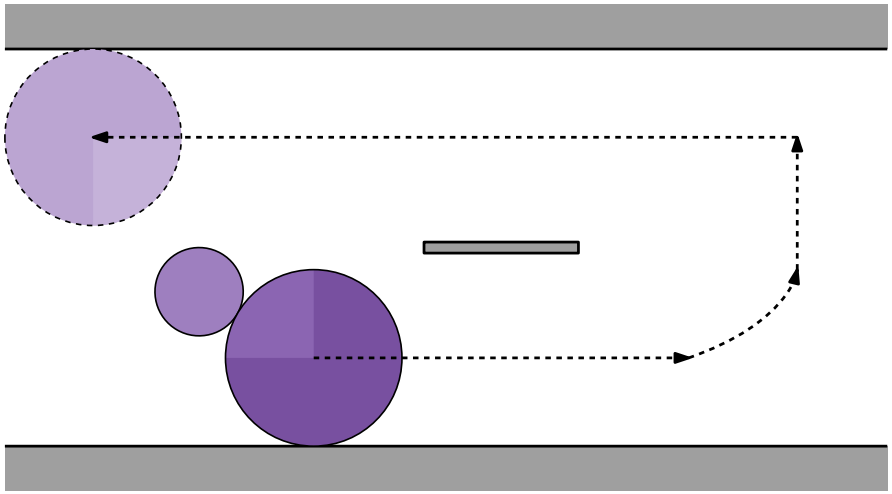
A configuration is also in the forbidden space if the pusher is outside the push range.

Forbidden push range also has $O(\#\text{sections})$ complexity.

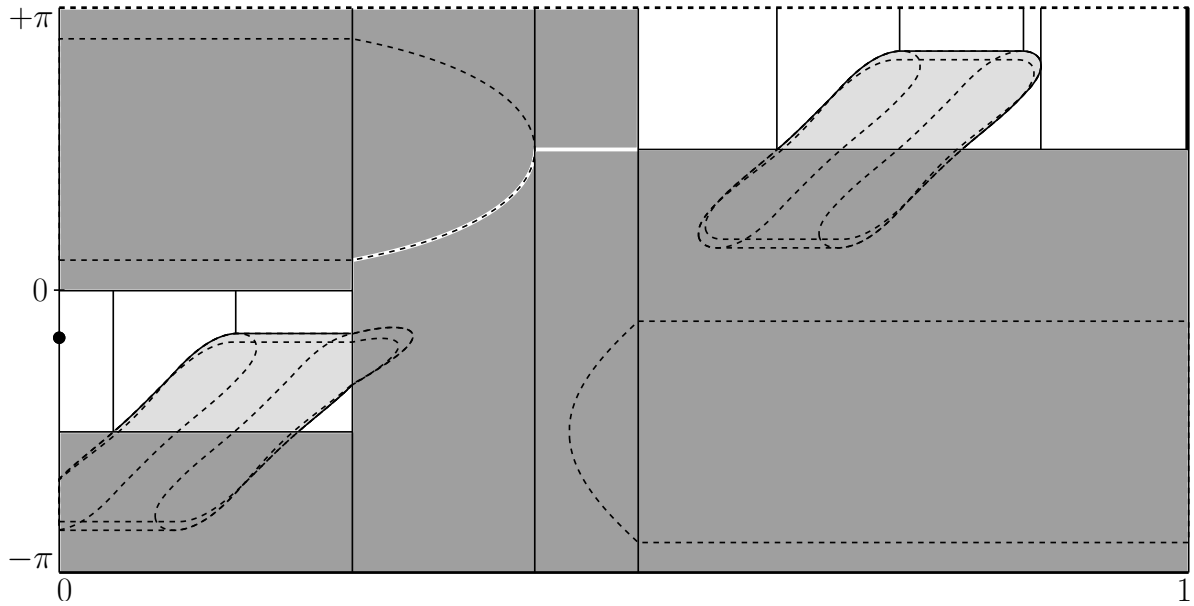
Total forbidden space: $O(\#\text{obstacles} \times \#\text{sections})$



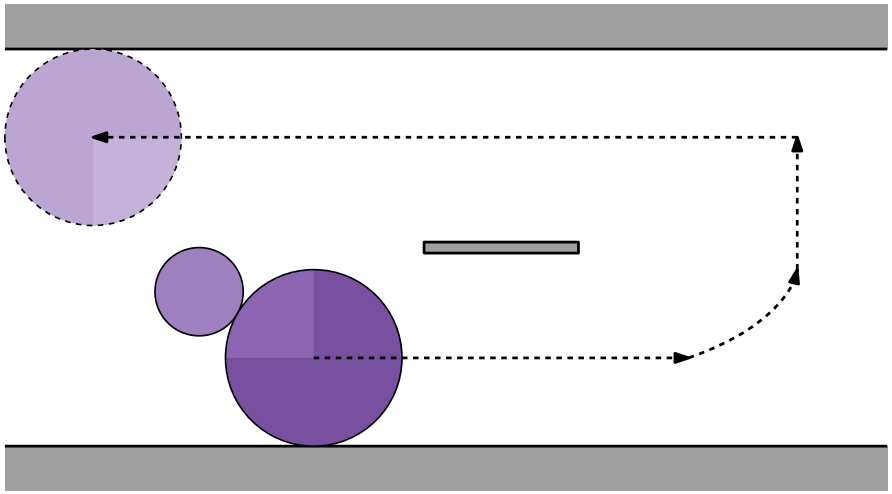
Finding a push plan



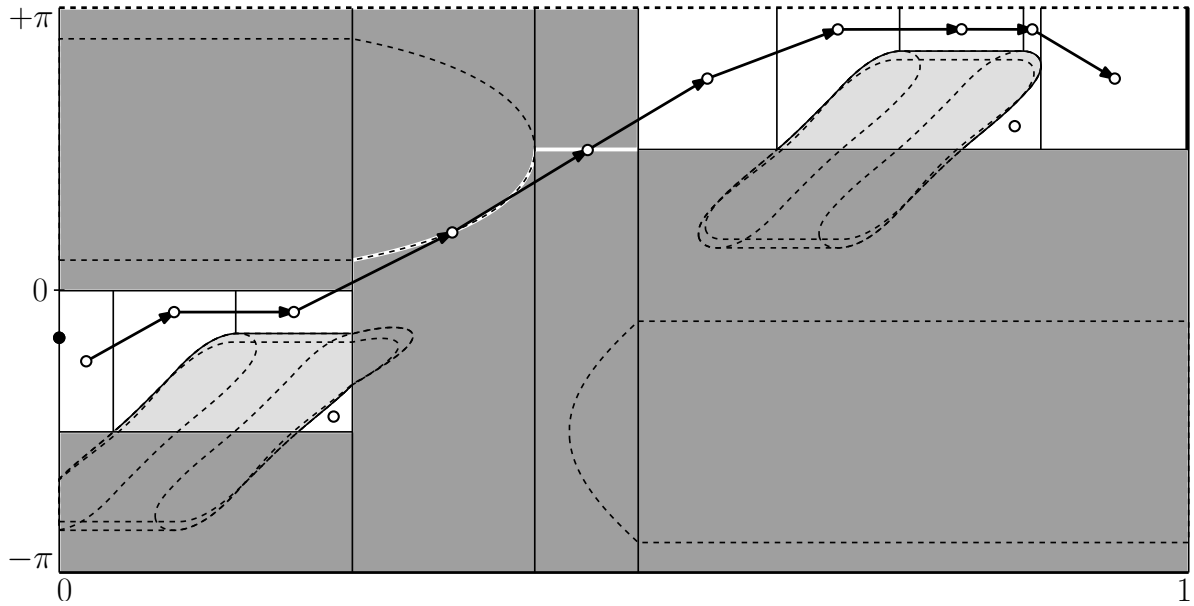
- Create vertical decomposition of the free space



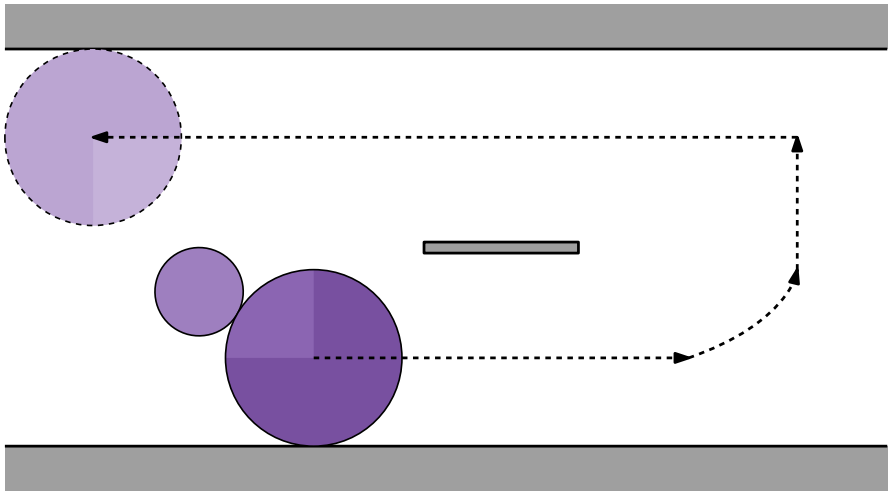
Finding a push plan



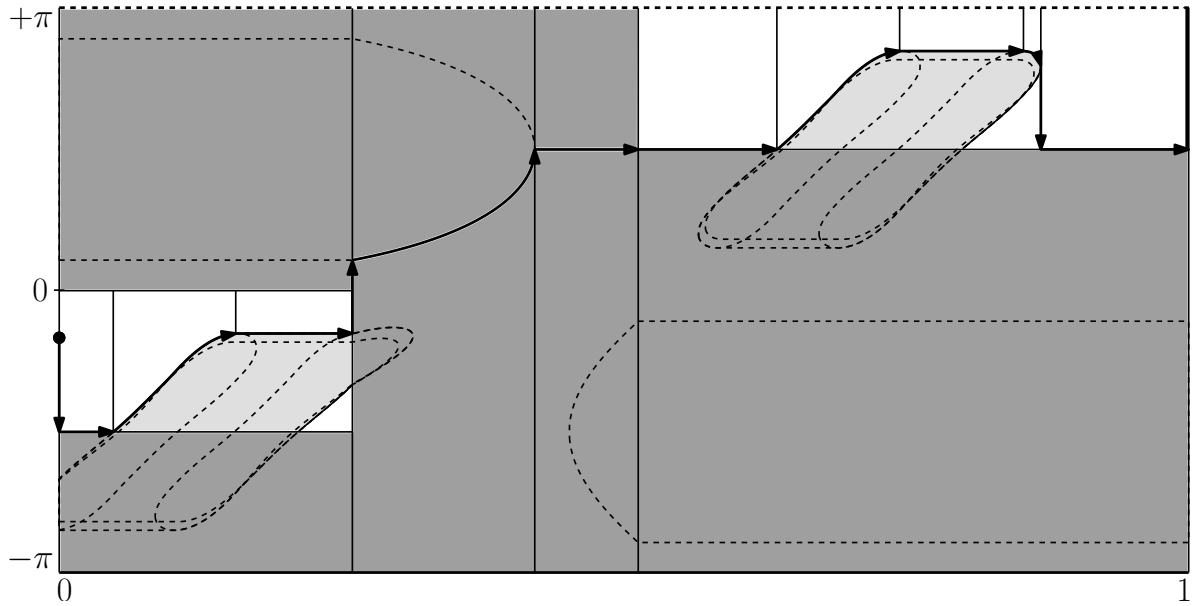
- Create vertical decomposition of the free space
- Find path through graph induced by cells



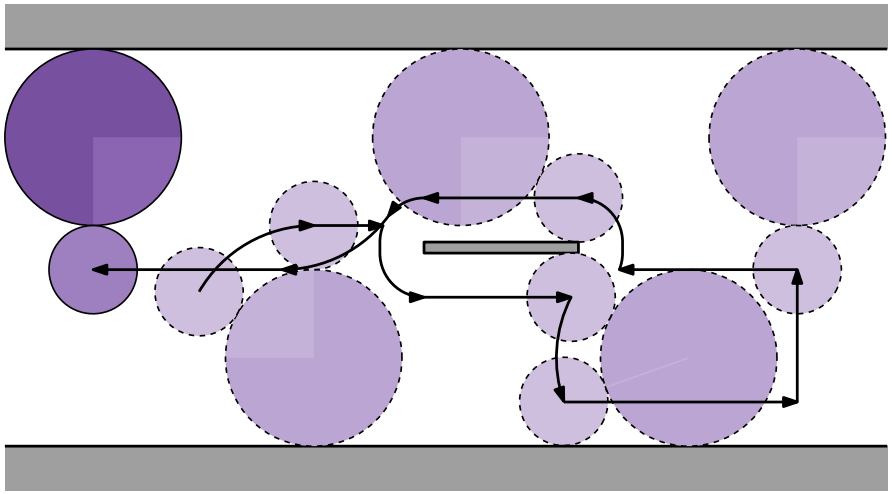
Finding a push plan



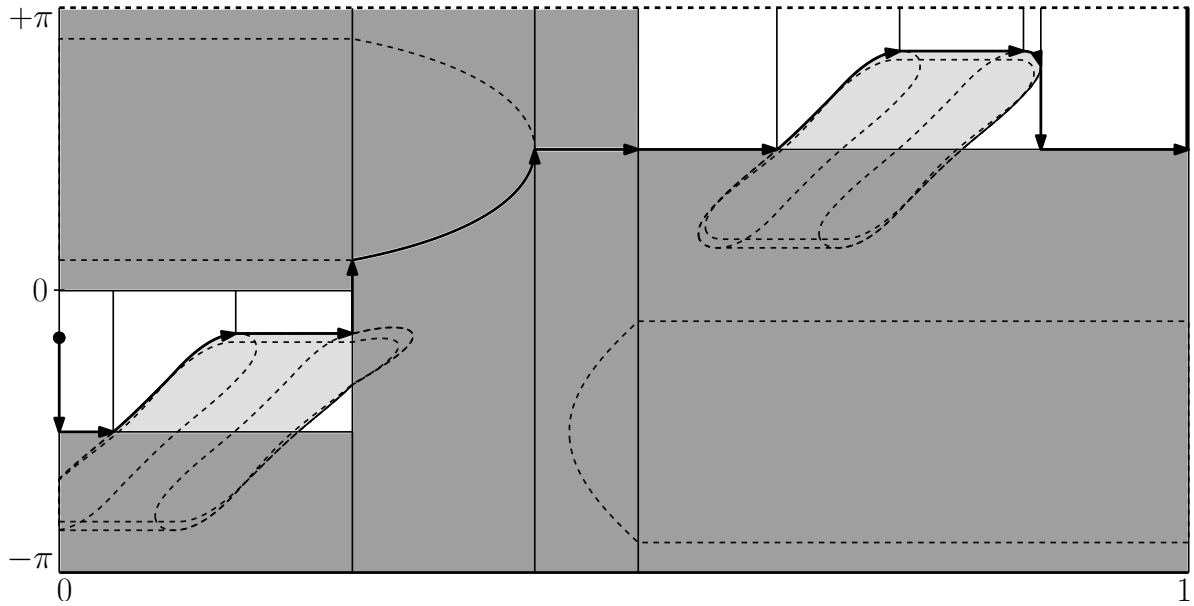
- Create vertical decomposition of the free space
- Find path through graph induced by cells
- Follow boundaries of those cells



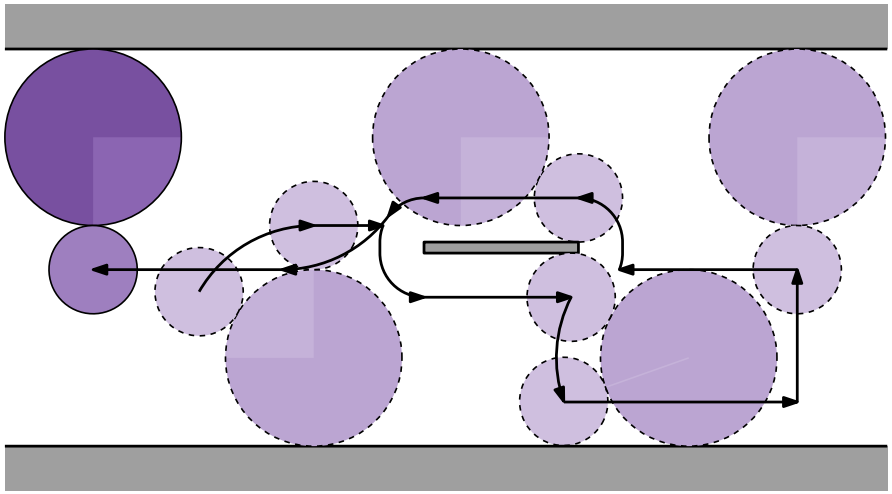
Finding a push plan



- Create vertical decomposition of the free space
- Find path through graph induced by cells
- Follow boundaries of those cells

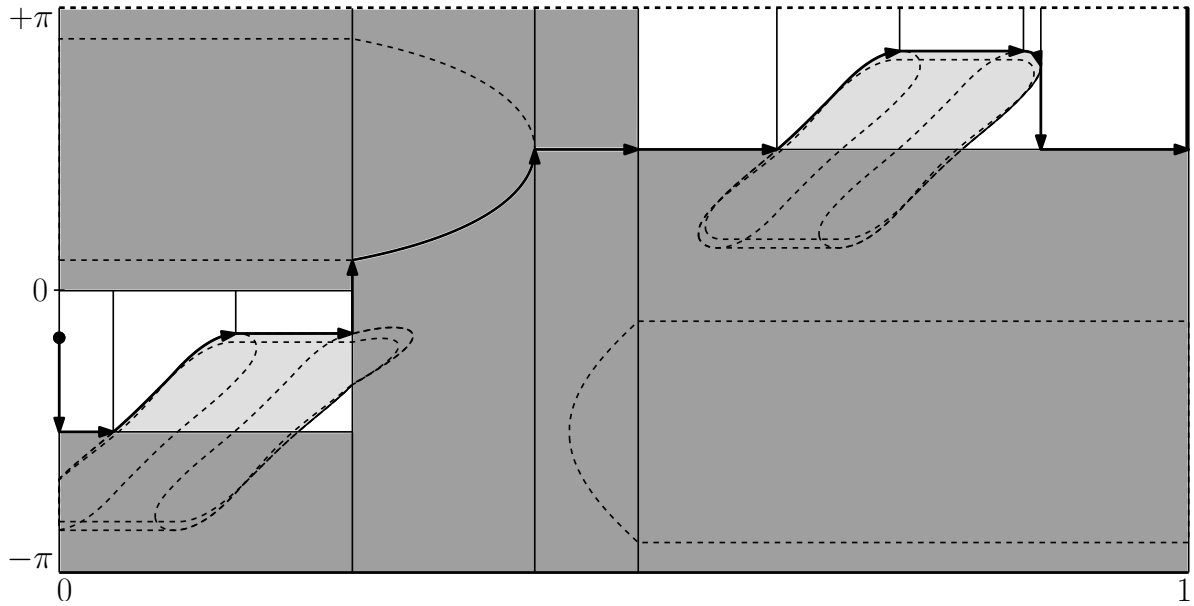


Finding a push plan

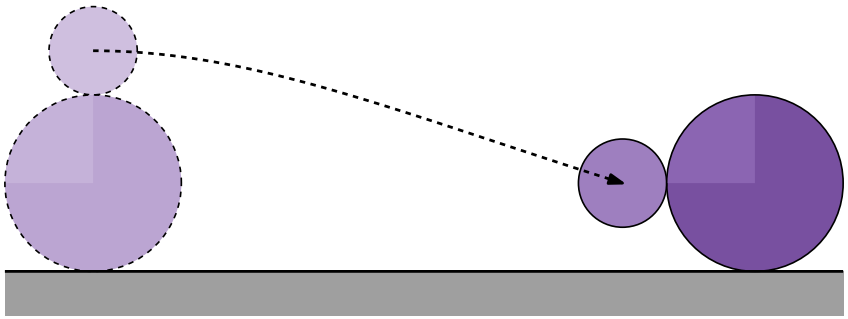


- Create vertical decomposition of the free space
- Find path through graph induced by cells
- Follow boundaries of those cells

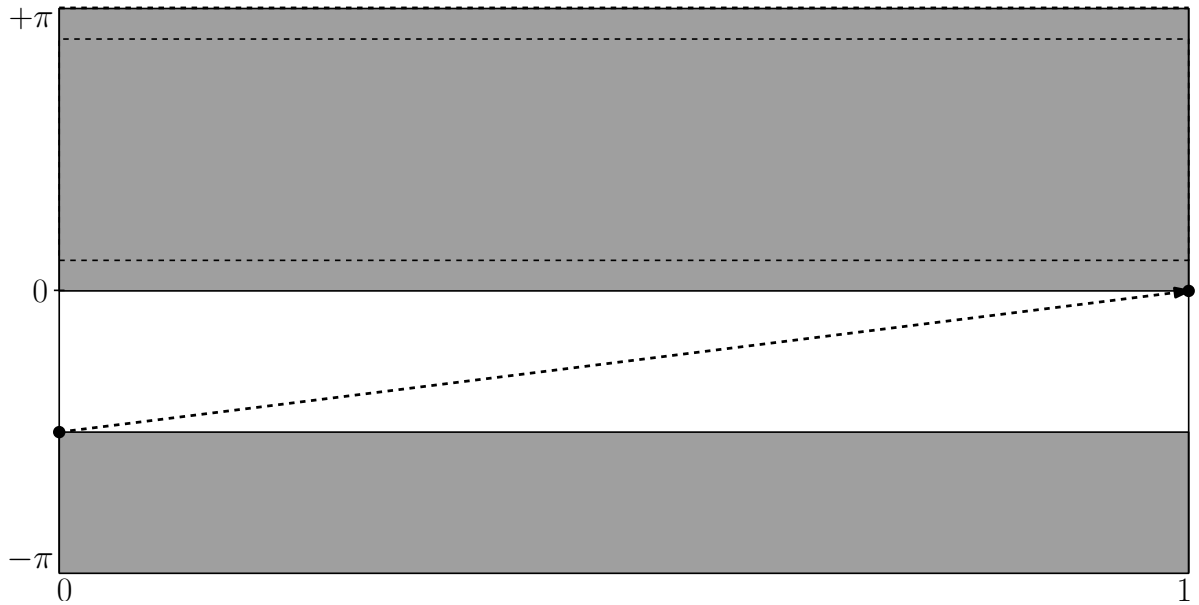
Fast method, but it gives a “bad” push plan. Can we find something shorter?



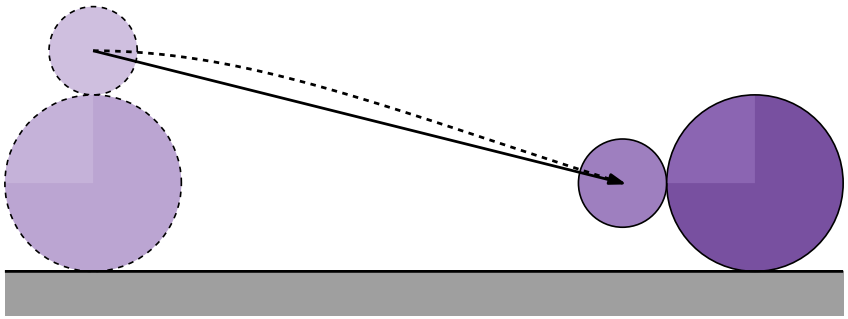
Finding a shortest push plan



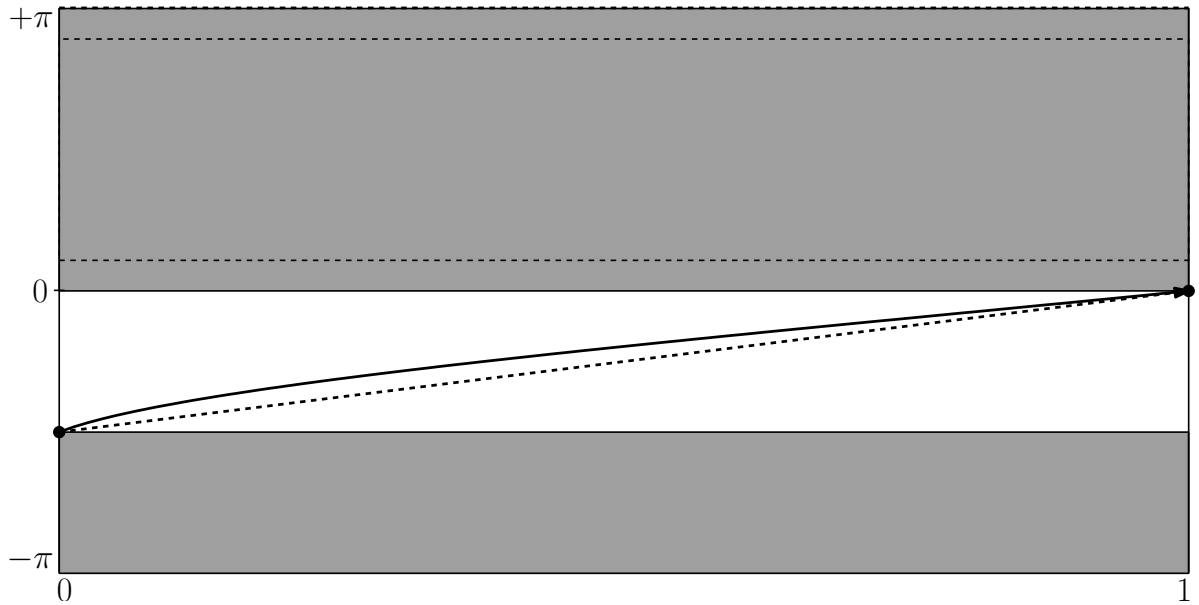
To minimize the distance traveled by the pusher, take the shortest path through free space?



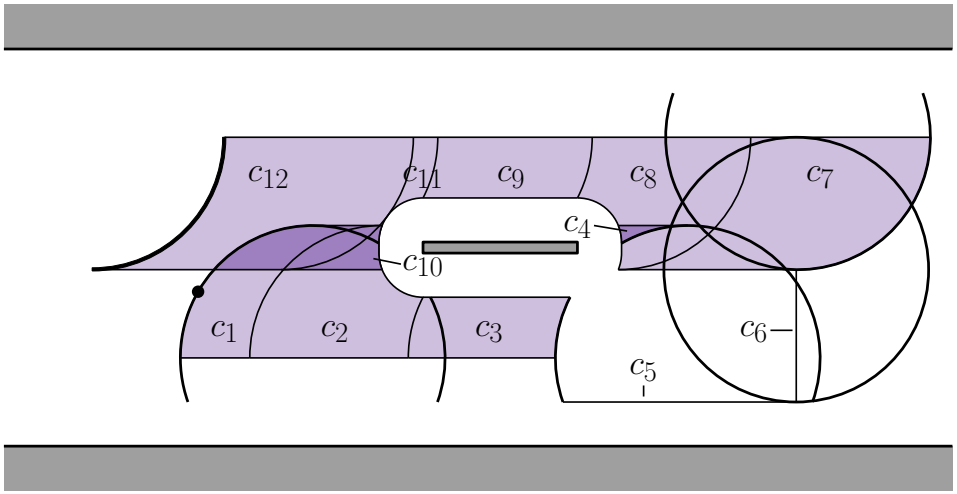
Finding a shortest push plan



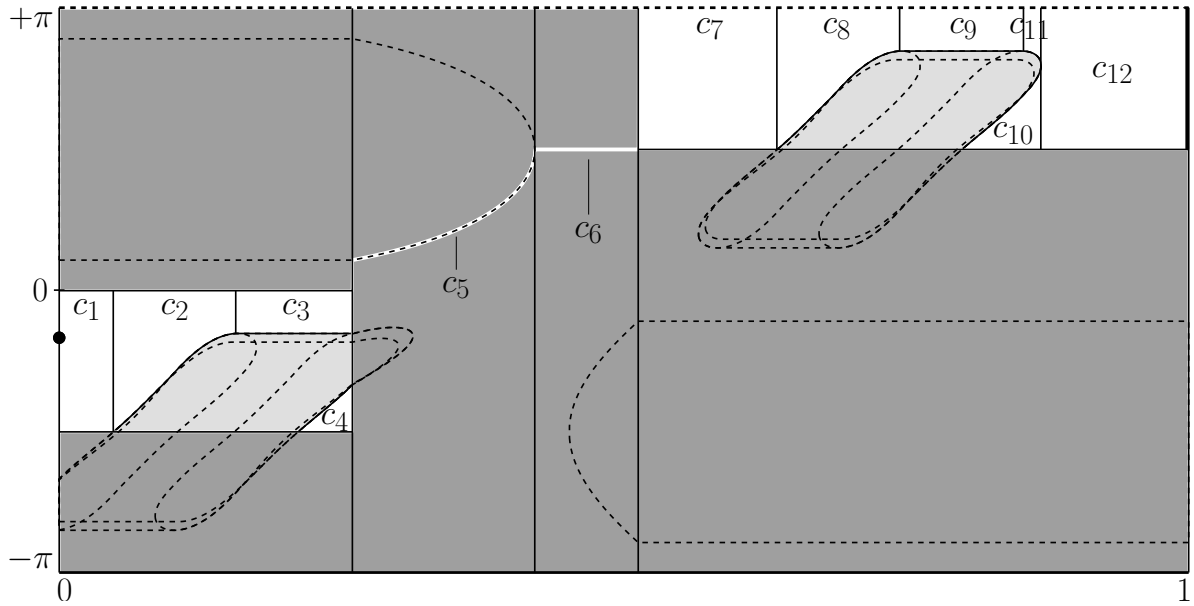
To minimize the distance traveled by the pusher, take the shortest path through free space? No!



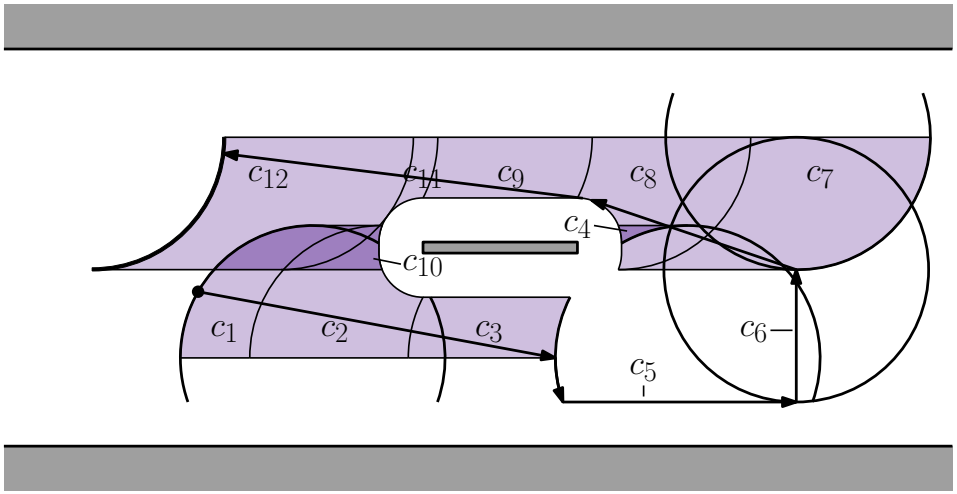
Finding a shortest push plan



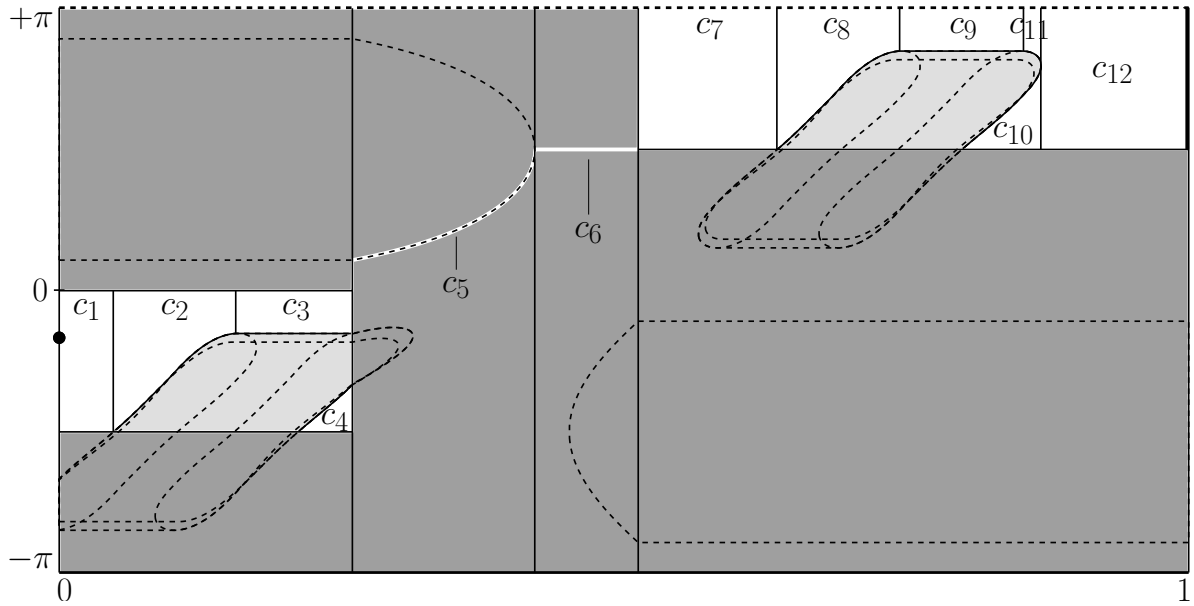
- Compute the area of allowed pusher positions for each cell



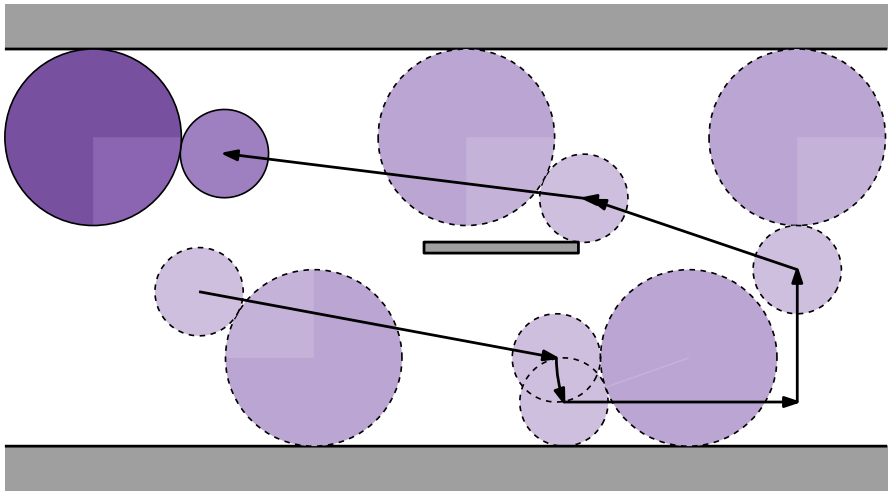
Finding a shortest push plan



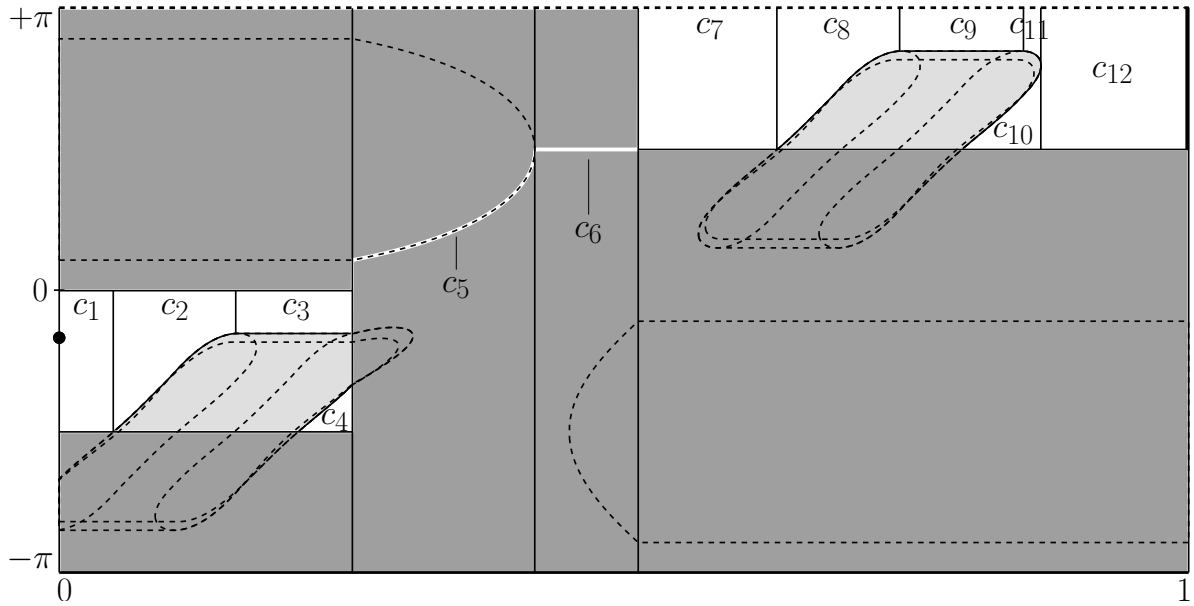
- Compute the area of allowed pusher positions for each cell
- Find a shortest path through these work-space cells



Finding a shortest push plan



- Compute the area of allowed pusher positions for each cell
- Find a shortest path through these work-space cells



New contributions

- Our algorithm for contact-preserving push plans is more general:
  - it can solve the pulling problem

New contributions

- Our algorithm for contact-preserving push plans is more general:
  - it can solve the pulling problem
  - it can handle non-line-segment obstacles

New contributions

- Our algorithm for contact-preserving push plans is more general:
  - it can solve the pulling problem
  - it can handle non-line-segment obstacles
  - it can handle a pusher larger than the object

New contributions

- Our algorithm for contact-preserving push plans is more general:
  - it can solve the pulling problem
  - it can handle non-line-segment obstacles
  - it can handle a pusher larger than the object

  while having a better running time!

New contributions

- Our algorithm for contact-preserving push plans is more general while having a better running time!

	High obstacle density		Low obstacle density	
	Nieuwenh.	Our method	Nieuwenhuisen	Our method
Preprocess	$n^2 \log n$	$n \log n$ (*)	$n^2 \log n$	$n \log n$ (*)
Any CPPP	$kn \log n$	$kn \log n$ (*)	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$

(*) These entries are expected times. For the worst-case times, replace $\log n$ by $\log^2 n$.

New contributions

- Our algorithm for contact-preserving push plans is more general while having a better running time!
- Can also compute shortest contact-preserving push plans

	High obstacle density		Low obstacle density	
	Nieuwenh.	Our method	Nieuwenhuisen	Our method
Preprocess	$n^2 \log n$	$n \log n$ (*)	$n^2 \log n$	$n \log n$ (*)
Any CPPP	$kn \log n$	$kn \log n$ (*)	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$
A shortest CPPP	—	$k^2 n^2 \log(kn)$	—	$(k+n)\log(k+n) + k^2 \log k$ (**)

(*) These entries are expected times. For the worst-case times, replace $\log n$ by $\log^2 n$.

(**) This yields a "quasi-optimal" solution.

New contributions

- Our algorithm for contact-preserving push plans is more general while having a better running time!
- Can also compute shortest contact-preserving push plans
- Can also compute unrestricted push plans, which we proved is sometimes necessary

	High obstacle density		Low obstacle density	
	Nieuwenh.	Our method	Nieuwenhuisen	Our method
Preprocess	$n^2 \log n$	$n \log n$ (*)	$n^2 \log n$	$n \log n$ (*)
Any CPPP	$kn \log n$	$kn \log n$ (*)	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$
A shortest CPPP	–	$k^2 n^2 \log(kn)$	–	$(k+n)\log(k+n) + k^2 \log k$ (**)
Any UPP	–	$kn \log(kn) + kn^2 \log n$	–	$(k+n)\log(k+n) + kn$

(*) These entries are expected times. For the worst-case times, replace $\log n$ by $\log^2 n$.

(**) This yields a "quasi-optimal" solution.

Further research

- Non-disk-shaped object and/or pusher

Further research

- Non-disk-shaped object and/or pusher
- Finding shortest unrestricted push plans

Further research

- Non-disk-shaped object and/or pusher
- Finding shortest unrestricted push plans
- Solving the global problem (i.e. no object path given) through the configuration-space method

Further research

- Non-disk-shaped object and/or pusher
- Finding shortest unrestricted push plans
- Solving the global problem (i.e. no object path given) through the configuration-space method
- Improved running times (especially for shortest push plans)

Questions?