Multirobot Pushing — How Many Robots are Sufficient?

Laura Lindzey Howie Choset Siddhartha S. Srinivasa
Ross A. Knepper

CMU-RI-TR-12-15

May 2012

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This paper asks how many cooperating homogeneous robots are required to perform a block pushing task in a known environment. This task is particularly challenging in the presence of a highly cluttered obstacle field where the connectivity of the robots' free configuration space depends on the block's configuration. In order to simplify the problem, we define an equivalence relation over block configurations based on the connectivity of the robots' free configuration space. We build a data structure that captures the relationships among the resulting equivalence classes, and then we encode constraints into the data structure that must be satisfied for the robots to be able to push the block between equivalence classes. We present an algorithm that operates on this data structure and uses existing optimization techniques to solve several variants of the minimum sufficient robots problem. Next, we give an implementation of this algorithm for an environment consisting of axis-aligned rectangles. Additionally, we provide a complete planner that finds a feasible path for the block in this environment.

Contents

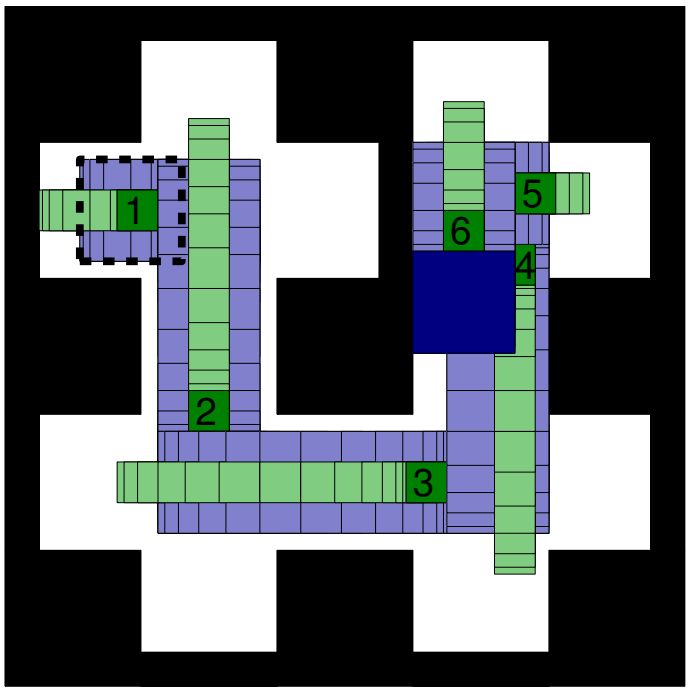


Figure 1: Example environment for block pushing. We show a block path (blue) that would require 6 robots (green), numbered in the order that they push the block. Intermediate robot and block positions are shown in lighter colors. A dashed square shows the initial block position.

1 Introduction

We are interested in problems where a group of robots work together to push a block from a starting configuration to a goal. Examples of this are shown in Figs. 1&2. This *multirobot block pushing* problem is challenging because in highly cluttered environments a path planner for the block must keep track of how robot positions affect the block's possible motions, and a path planner for the robots must look ahead and reason about which connected component each robot needs to occupy as the block's motion partitions the robots' free space.

While other researchers have looked at how to plan for a team of robots pushing a block, the focus of this paper is not an algorithm to do that. Instead, we derive a bound on how many robots are required to push the block along any feasible path in a given environment. The resulting bound applies to the solution found by any planning algorithm. Consider the environment shown in Fig. 1. Whereas the illustrated block path only requires six robots, there exist paths that require up to nine robots.

This paper provides a solution to the following problems:

- **P1**: Given an environment with static obstacles and shapes for the block and

robots, determine the minimum number of robots required to push the block along *any* feasible path in the environment.

- **P2**: Given the same setup as above and a *known* initial block configuration q_0, determine the minimum number of robots required to push the block along any feasible path starting at q_0. This may require fewer robots than **P1** because there may exist portions of the environment that are not reachable from every initial position.

We then provide an implementation that solves these problems in an environment with axis-aligned rectangles, and we solve the block planning problem in this environment.

We are able to solve these minimum sufficient robots problems by reformulating block pushing as a constrained minimization problem with constraints derived from two properties of the environment. First, we require that robots obey the semantics of pushing, which we term *manipulability*.[1] Next, as the block's motion changes the connectivity of the robots' free space, we require that each robot must move consistently between merging and splitting connected components. We call this *conservation of robots*. These properties induce constraints on the number of robots occupying each connected component. We provide a bound on the minimum sufficient robots needed in any environment for which we can enumerate these constraints. By grouping constraints based on which pushes are feasible in tight spaces, we can guarantee a tight bound.

2 Related Work

Approaches to manipulation planning often consider a set of alternating navigation and manipulation actions. This makes it difficult to apply typical motion planning algorithms. Under some conditions, the manipulation planning problem can be split into two steps: first choosing a path for the block, and then finding robot paths that cause the block to follow that path [16]. This decomposition is similar to Koga and Latombe's [5] division of multi-arm manipulation problems into *transit* and *transfer* tasks. Our work focuses exclusively on what is required to find a feasible block path, instead of on solving the navigation problem for individual robots. Once a path for the block has been found, it imposes a set of constraints on individual robot positions and motions. Robot paths obeying these constraints can be found using existing multirobot planning algorithms [4, 15].

Significant previous work has focused on the mechanics of block pushing and the problem of how a team of robots can cause a block to follow a predetermined path. Lynch and Mason investigated the controllability of point- and line-contact pushing [7], [8]. More recently, de Berg and Gerrits [2] investigated how to compute paths for a team of robots to push a block along a given path among obstacles. Early work on the robot cooperation facet of this problem includes a demonstration that two robots are able to complete an obstacle free block pushing task more efficiently than a single robot [9] and an examination of how a group of robots can reorient blocks [11].

[1] This is distinct from Yoshikawa's manipulability for kinematic chains [17]. It is more closely related, but not equivalent, to the concepts of *accessibility* and *controllability* [1].

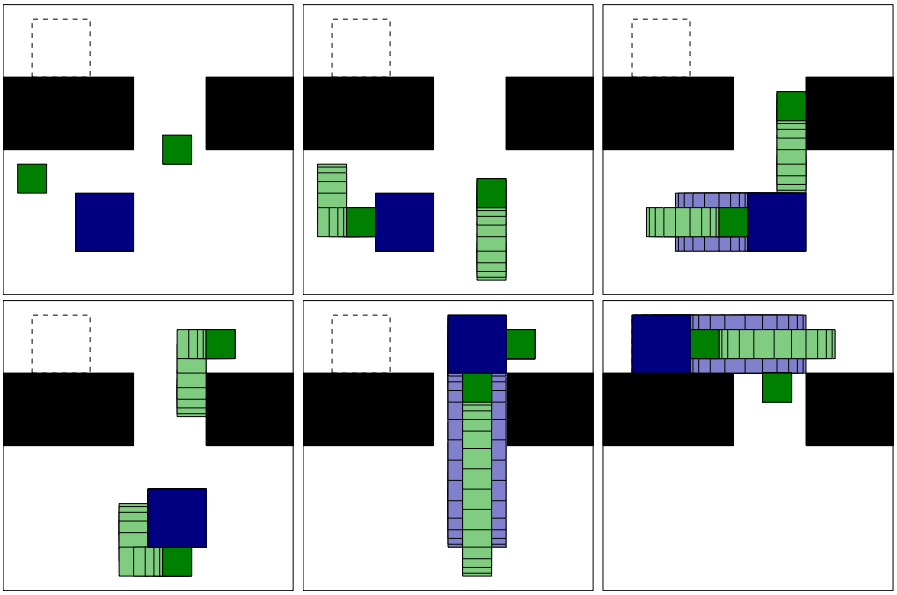


Figure 2: Example block pushing problem. The robots (green squares) must push the movable block (blue square) to the goal configuration (dotted outline square), in the presence of static obstacles (black rectangles). Lighter blue and green shows the history of block and robot motions.

Caging is another method for solving the multirobot block pushing problem. Rather than alternating navigation and manipulation actions, robot actions are chosen such that they approach the goal while obeying constraints guaranteeing that the block remain caged. This approach has resulted in complete algorithms for obstacle free environments [14], and moderately cluttered environments [13, 10, 3]. However, we consider environments with narrow passages where it is not physically possible to cage an object.

3 Definitions and Problem Statement

Throughout this paper, we work in a closed, bounded subset of $\mathbb{R}^2$, populated by obstacles $\mathbf{O} = \{O_i\}$. Identical robots $\mathbf{R} = \{R_i\}$ cooperate to manipulate the block B, and are able to perform two types of actions within this environment: *navigation actions*, where they move within a connected component of their free configuration space; and *manipulation actions*, where they maneuver B by pushing, but not pulling. A solution for the block pushing problem consists of a block path from a start configuration x_s to a goal configuration x_g and a set of robot trajectories $P_{\mathbf{R}} = \{P_{R1}, P_{R2}, \dots\}$ that cooperate to push B along the path.

Q_B^{free} is the free configuration space formed by the block B moving among obstacles $\mathbf{O}$. Let the continuous function $P_B : [0, 1] \rightarrow Q_B^{free}$ be a path for the block, and

DEFINITIONS

$\mathbf{O} = \{O_i\}$	obstacles
$\mathbf{R} = \{R_i\}$	robots
B	manipulated block
$Q_R^{free}(x)$	free configuration space of robot R with B at x
Q_B^{free}	free configuration space of B
$N(Q)$	number of connected components in space Q
$N(x)$	shorthand for $N\left(Q_R^{free}(x)\right)$ for $x \in Q_B^{free}$
P_{Ri}	path of R_i
$P_{\mathbf{R}} = \{P_{R1}, ..., P_{Rn}\}$	set of robot paths
P_B, $\mathbf{P}_B$	path for B, set of all such paths
F_B, $\mathbf{F}_B$	feasible path for B, set of all such paths
EC	Equivalence Class
EG	Equivalence Graph
PG	Precursor Graph
RG	Reachability Graph
α_i	ith connected component of EC α
$m_{\alpha i}$	number of robots occupying α_i

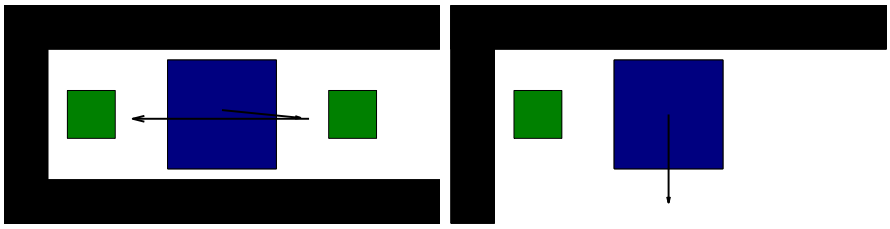
the set of all such paths be $\mathbf{P}_B$. We define *feasible paths* to be the subset of block paths that the robots are able push the block along:

$$\mathbf{F}_B = \{p \in \mathbf{P}_B \mid \exists P_{\mathbf{R}} \text{ causing } B \text{ to follow } p\}$$

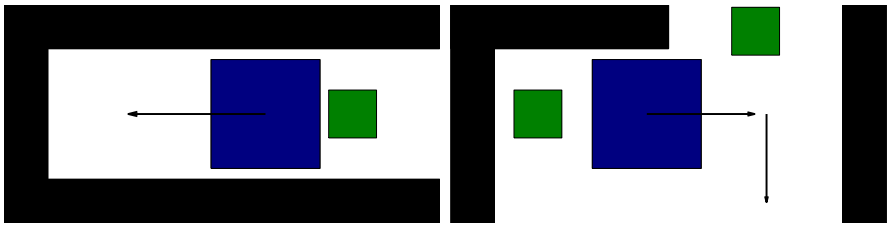
Some paths are infeasible because robots occupy finite volume and because robots can only push and not pull on the block (Fig. 3).

We assume that all of the robots are the same shape and therefore are only required to keep track of a single configuration space for the robots. We define $Q_R^{free}(x)$ to be the free configuration space formed by any robot R_i moving among obstacles $\mathbf{O}$ with the block at position x. Additionally, we require $Q_R^{free}(\emptyset)$, the robots' free configuration space in the absence of B, to have a single connected component. When analyzing block configurations, it is convenient to specify the number of connected components of Q_B^{free}. We define a function $N(Q)$ that returns the number of connected components in a configuration space Q.

In order to tractably reason about all possible block paths, we define an equivalence relation on block positions x, such that any path can be broken down into a series of actions transitioning among equivalence classes (ECs). We say that two block configurations x_i and x_j are equivalent if there exists a continuous path $p \in \mathbf{P}_B$ parametrized by $s \in [0, 1]$ with $p(0) = x_i$ and $p(1) = x_j$ along which $N\left(p(s)\right)$ is held constant. Each EC α is composed of a set of connected components $\{\alpha_1, \alpha_2, \ldots\}$, as shown in Fig. 6. Finally, we use $m_{\alpha i}$ to represent the number of robots occupying the connected component α_i.



a Two infeasible scenarios: (left) occupied connected component of Q_R^{free} disappears, and (right) no robot can access the top face of B to push it down.



b Two feasible scenarios: (left) connected component to left of B is not required to be occupied, and (right) robot can access the top face of B to push it down.

Figure 3: Examples of feasible and infeasible paths.

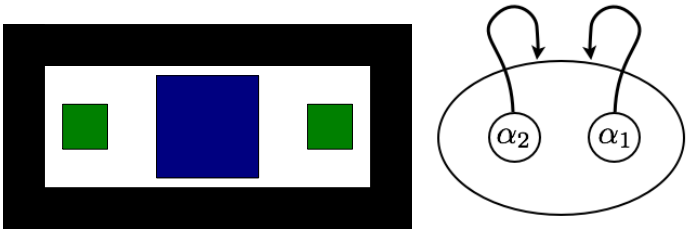


Figure 4: Example of self-loop edges. The block can move left and right, but can never leave the EC. $Q_R^{free}(x)$ has two connected components, each occupied by one robot. Motion within the EC is represented by self-loop edges starting at each nodelet and terminating at the same EC.

4 Approach

In this section, we describe the data structures that we use to encode the problem and present Algorithm 1, which solves the minimum sufficient robots problems. In the following section, we show how our algorithm can be implemented in one class of environment.

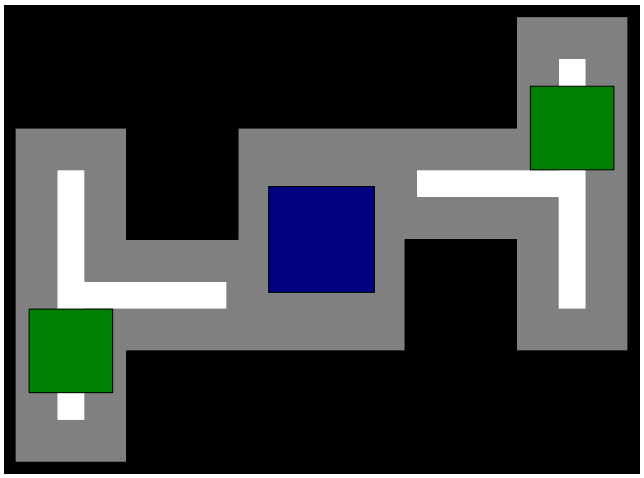


Figure 5: Example environment where using the set of all constraints would give an overestimate of the minimum sufficient robots.

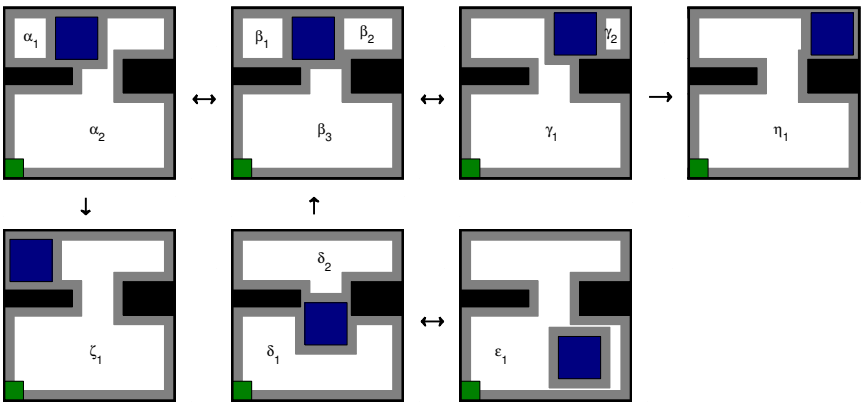


Figure 6: An EG where each node is represented by an example block configuration. Transitions between nodes are shown by arrows, and nodelets are given alphanumeric labels. We have omitted the self-loop arrows. In each configuration, we overlay the workspace obstacles (black), block position (blue) and robot (green) on the robots' C-obstacles (gray).

4.1 Equivalence Graph

The *equivalence graph* (EG) encodes a compact representation of the topology of the environment and the motion of the block. The EG is a graph where each node corresponds to an equivalence class (EC). Each EC contains a number of connected components, which are represented by *nodelets* associated with the node, as shown in Fig. 7. Directed edges represent feasible block motions: an edge from α_1 to β indicates

that there exists at least one block configuration within α for which a robot in connected component α_1 can push the block to EC β. However, it does not guarantee that the transition is feasible from all configurations within α. Self-loop edges correspond to block motions that do not result in the block transitioning to a new EC. Consider the environment shown in Fig. 4, which only has one EC. If there is a robot in each connected component of the EC, then they can push the block back and forth. This is represented by the self-loop edges. For each edge on the EG, we compute the corresponding manipulability and conservation of robots constraints.

Every edge in the EG corresponds to a potentially feasible block motion, and a block path can be described by a corresponding walk through the EG. Thus, we have the necessary condition that for a feasible block path to exist, the corresponding walk in the graph must exist. However, this is not a sufficient condition, because the EG's edges represent that a transition is feasible from at least one configuration within the originating EC but do not guarantee that it is possible from all configurations. In order to guarantee that a feasible path can be followed, we need an assignment of robots to connected components that is consistent with the constraints associated with every EG transition on the path.

4.2 Reachability Graph

It is not guaranteed that all transitions between ECs can be reached from a given starting point. For example, consider the environment shown in Fig. 5. There is no initial block position from which it is possible to reach both the left and the right sections of the environment. This means that considering the constraints associated with every transition in the EG will provide an overestimate on how many robots are required. Thus, in order to find a tight bound, we need to determine maximal sets of transitions that can be reached from a single block configuration, and then consider only their associated constraints.

The *reachability graph* (RG) encodes all feasible ways to chain together transitions between ECs. It has two key properties: any feasible block motion must map to a walk on the RG, and for any walk on the RG we must be able to determine which EC transitions have been crossed. We use this information to determine which constraints are mutually applicable. If each edge represents a locally feasible block motion, then a transitive closure of RG edges exactly indicates a maximal set of feasible block motions within the environment. For each maximal set of block motions, we determine the EC transitions involved and how many robots are required by their associated constraints.

4.3 Algorithm Overview

As the first step in calculating the EG, we need to identify all of the environment's ECs (Algorithm 1, line 1). We do not necessarily need to find an explicit division of Q_B^{free} into ECs: for our purposes, it is sufficient to identify the set of all ECs and transitions among them.

Next, we construct the EG (Algorithm 1, line 2). We create a node in the EG for each EC calculated in the previous step. The EG's edges are derived from feasible transitions among ECs. Additionally, for each EC, we compute $N(x)$, for any block

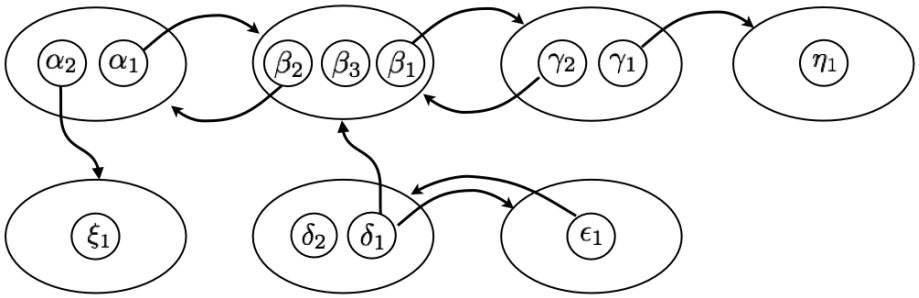


Figure 7: Symbolic representation of the same EG as Fig.6. Nodes are ovals, containing their corresponding nodelets. Transitions are shown as arrows, originating at the nodelet that must be occupied for the transition to be feasible and terminating at the resulting EC. Self-loops have been omitted.

configuration x within that EC. We represent each connected component as a nodelet within the corresponding node of the equivalence graph.

The final step in constructing the EG is to associate constraints with each transition (Algorithm 1, line 4). Recall the manipulability and conservation of robots constraints described in Section 1. We now see that these are constraints on the number of robots assigned to each nodelet. For example, in the environment shown in Fig. 6, consider a transition from EC δ to β. It imposes the following constraints:

$$\begin{aligned} m_{\delta 1} &= m_{\beta 3} && \text{(conservation of robots)} \\ m_{\delta 2} &= m_{\beta 1} + m_{\beta 2} && \text{(conservation of robots)} \\ m_{\delta 1} &\geq 1 && \text{(manipulability)}. \end{aligned}$$

The constraints on $m_{\alpha i}$ guarantee that robots will be positioned to enable every feasible block transition. Given a set of such constraints, we determine the minimum sufficient number of robots by assigning robots to connected components in a way that minimizes the total number of robots in any given EC while satisfying all constraints (Algorithm 1, lines 6 and 7). This is an integer constrained minimization problem with bounded variables.

When generating a list of constraints, we ignore transitions to a sink node in the EG where a connected component of $Q_R^{free}(\cdot)$ vanishes without merging with another. See Fig. 6 (nodes η and ζ) for examples of sink nodes. A disappearing connected component imposes a constraint of the form $m_{\alpha 1} = 0$, specifying that the connected component is required to not be occupied. This creates a conflict if the same connected component is required to be occupied by another manipulability constraint, as is the case for a sink. We resolve the conflict by ignoring the constraint and the sink node. This does not affect the number of robots required. A block path without the transition to a sink cannot require fewer robots than a path that stops one transition before, and adding an additional upper limit on number of robots can't increase the number required.

Algorithm 1 getMinimumSufficientRobots

Input: $\mathbf{O}, B, \mathbf{R}$
Output: $minNum$
1: $ECs \leftarrow getEquivClasses(\mathbf{O}, B, \mathbf{R})$
2: $EG \leftarrow getEquivGraph(ECs)$
3: $RG \leftarrow getReachabilityGraph(EG, \mathbf{O}, B, \mathbf{R})$
4: $constraints \leftarrow getConstraints(nodes, edges, nodelets)$
5: $constraintSets \leftarrow getReachabilityConstraints(RG, EG)$
6: $\{m_{\alpha i}, m_{\beta i}, \dots\} \leftarrow solveConstraints(constraintSets)$
7: $minNum \leftarrow \sum_i numRobots(m_{\alpha i})$

5 Implementation

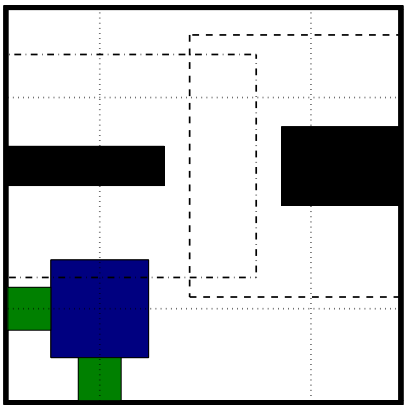
Our choice of environment is motivated by an attempt to provide as simple an example as possible while still retaining the complex configuration space structure that we are interested in. All objects in our environment ($\mathbf{O}$, B, and $\mathbf{R}$) are closed, axis-aligned rectangles. Surface contact, including sliding, is allowed between any pair of objects, but the intersection of their interiors must be empty. The robots $\mathbf{R}$ may translate freely within their respective connected components of $Q_R^{free}(x)$, but rotation is forbidden. All motion of the block B is aligned with an axis and is generated by a single robot R_i pushing B, in face-to-face contact. The resulting $B + R_i$ assembly can only move in the direction of B's inward-pointing contact normal.

We reduce the problem of partitioning Q_B^{free} into ECs to that of classifying a set of *candidate points*. Candidate points are sampled in such a way as to guarantee that at least one point is chosen from each EC. They also need to capture all possible transitions between the ECs. In order to do this, we include endpoints of at least one representative block motion that crosses each inter-EC boundary. We introduce an intermediate data structure called the *precursor graph* (PG) which we use to group the candidate points into ECs and to determine feasible transitions for the EG. The PG is an undirected graph with a node corresponding to each candidate point in Q_B^{free} and edges corresponding to block motions between the candidate points. We also use the PG to generate the *reachability graph* (RG), which is a directed graph that encodes information about the feasibility of block motion.

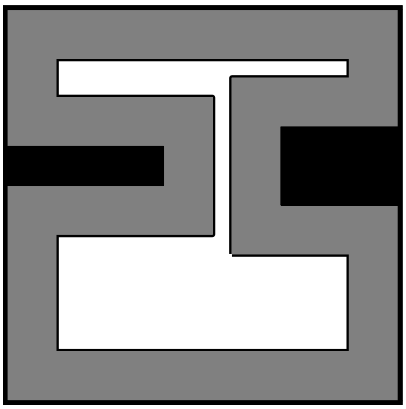
5.1 Enumeration of Equivalence Classes

Our first task is to divide Q_B^{free} into ECs. For the axis-aligned rectangle world, this division into ECs can be performed exactly without explicitly testing every point $x \in Q_B^{free}$. We construct a rectangular tiling of the world where any possible EC boundary is coincident with a tile's boundary. We then use the tiling to build the precursor graph and group PG nodes into ECs.

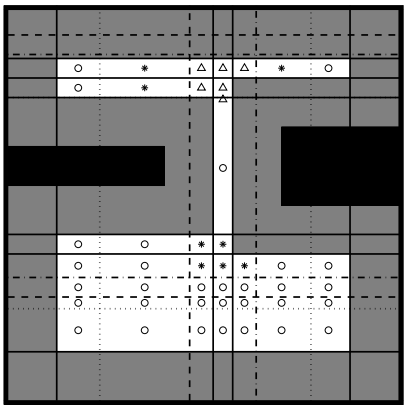
There are two types of EC boundaries: those imposed by the boundaries of Q_B^{free} and those created by transitions between ECs. For our axis-aligned environment, the boundaries of Q_B^{free} are a set of vertical and horizontal line segments. Transitions between ECs can only occur when the block's edge is exactly a robot width or height



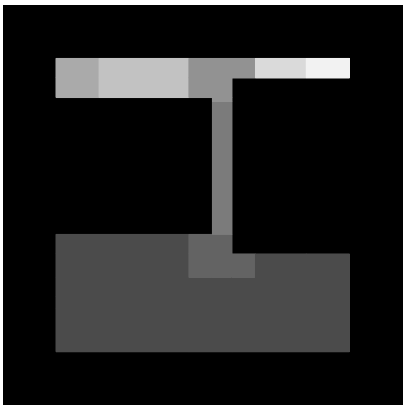
a Potential EC boundaries derived from transitions: For the given obstacles (black), block (blue) and robots (green), the boundaries corresponding to the environment's edge are dotted, the ones for the left obstacle are dash-dotted, and for the right obstacle are dashed.



b Potential EC boundaries derived from Q_B^{free}: For the given obstacles (black), we show the C-obstacle (gray) and its boundary line segments (solid lines)



c Division of Q_B into tiles. Lines are drawn in the same style as the line segment from a or b that they are extending. Tiles in Q_B^{free} are labeled with $N(x)$: $\triangle$ for 3, $*$ for 2, $\circ$ for 1.



d Division of Q_B^{free} into ECs. Black indicates C-space obstacle, and each different shade of grey is a different EC.

Figure 8: Calculation of Equivalence Classes

away from an obstacle. This is because, by definition, any block displacement Δx that results in the block crossing one EC boundary requires that $N(x) \neq N(x + \Delta x)$. This corresponds to "pinching off" or "opening up" a previously (im)passable corridor for the robot. We show an example of potential EC boundaries for all obstacles in an environment in Fig. 8a.

We now have a set of horizontal and vertical line segments that are guaranteed to cover every EC boundary. In order to generate a rectangular tiling of Q_B^{free} we extend each line segment to span the environment, as shown in Fig. 8c. The resulting tiling informs our choice of candidate points.

Algorithm 2 getEquivClasses

$minX(O)$ and $maxX(O)$ are the minimum and maximum coordinates of O. B_x and B_y are the B's width and height, and R_x and R_y are R_i's dimensions.

Input: $\mathbf{O}, B, \mathbf{R}$
Output: $PG = (N_p, E_p), ECs$

// Part 1: lists of x and y coordinates for tile boundaries
$x, y \leftarrow \emptyset$
for all $O \in \mathbf{O}$ **do**
  // add C-space boundary coordinates
  $x \leftarrow x \cup \{minX(O) - \frac{1}{2}B_x, maxX(O) + \frac{1}{2}B_x\}$
  $y \leftarrow y \cup \{minY(O) - \frac{1}{2}B_y, maxY(O) + \frac{1}{2}B_y\}$
  // add potential EC transition coordinates
  $x \leftarrow x \cup \{minX(O) - R_x - \frac{1}{2}B_x, maxX(O) + R_x + \frac{1}{2}B_x\}$
  $y \leftarrow y \cup \{minY(O) - R_y - \frac{1}{2}B_y, maxY(O) + R_y + \frac{1}{2}B_y\}$
$PG = (N_p, E_p) \leftarrow (\emptyset, \emptyset)$
for all tiles defined by x and y lines **do**
  $N_p \leftarrow N_p \cup \{$tile's centroid, edge midpoints, corners$\}$
  $E_c \leftarrow E_c \cup \{$centroid$\leftrightarrow$midpoint, midpoint$\leftrightarrow$corner$\}$
// Part 2: remove nodes in collision with obstacles
for all $n \in N_p$ **do**
  if $n \notin Q_B^{free}$ **then**
    $N_p = N_p \backslash n$
    $E_p = E_p \backslash \{$all e with n as endpoint$\}$
  else
    $n.label \leftarrow N(Q_R^{free}(n))$
// Part 3: group candidate points based on $N(x)$
$numEC \leftarrow 0$
while $\exists n \in N_p | n.ec = \emptyset$ **do**
  $S \leftarrow \{n\}$
  $numEC \leftarrow numEC + 1$
  $n.ec \leftarrow numEC$
  while $S \neq \emptyset$ **do**
    $u \leftarrow$ an element from S
    $S \leftarrow S \setminus \{u\}$
    for all $v \in N_p \mid (u, v) \in E_p$ **do**
      if $v.ec = \emptyset \wedge u.label = v.label$ **then**
        $v.ec \leftarrow numEC$
        $S \leftarrow S \cup \{v\}$

Recall that the PG's nodes are required to provide an oversampling of both ECs and endpoints of feasible transitions among ECs. We guarantee that every EC is sampled by adding the centroid of each tile to the list of candidate points. We additionally need to guarantee that we have included start and end points for any feasible EC transition. Any feasible block motion must allow space for the pushing robot behind the block throughout the push. The feasible start and end points are exactly the same as the coordinates that were used to find potential EC boundaries. Thus, we guarantee that any feasible transition is represented by adding the corners of every tile and the midpoints

Algorithm 3 getReachabilityGraph

Input: $PG = (N_p, E_p)$
Output: $RG = (N_r, E_r)$
1: $(N_r, E_r) \leftarrow (N_p, \emptyset)$
2: **for all** $e \in E_p \mid e = (p_1, p_2)$ **do**
3: **if** $p_1 \rightarrow p_2$ is feasible **then**
4: $E_R \leftarrow E_r \cup \{(p_1, p_2)\}$
5: **if** $p_2 \rightarrow p_1$ is feasible **then**
6: $E_r \leftarrow E_r \cup \{(p_2, p_1)\}$
7: **return** RG

of all four bounding line segments to the list of candidate points. The PG's edges are formed by connecting candidate points corresponding to edge midpoints to those corresponding to the centroids of the same tile, as well as connecting corners to edge midpoints.

The final step in partitioning Q_B^{free} into ECs is grouping the PG's nodes. We calculate $N(x)$ for every node in the precursor graph, where x is the node's workspace coordinate. We are then able to generate a labeling of ECs by grouping all nodes x_i, x_j such that $N(x_i) = N(x_j)$ and $[x_i, x_j]$ is an edge in the PG. Each of these groups corresponds to a single EC. An exact partition of Q_B^{free} into ECs is shown in Fig. 8d.

5.2 Reachability Graph Generation

In this environment, the PG's nodes provide a sufficient set of nodes for the RG—they include coordinates in each EC, and due to the construction of the PG, if a feasible transition between two nodes exists, there will also be one that follows the edges of the EG. However, the PG is an undirected graph encoding adjacency, while the RG needs to be a directed graph describing reachability. Thus, we test both possible motion directions for each undirected edge in the PG, and create a directed edge in the RG for every feasible transition. According to our motion model, a block transition from configuration x_1 to x_2 is *feasible* if the block does not collide with an obstacle and if a robot can reach the proper side of the block to execute the push.

5.3 Equivalence Graph Generation

First, we need to find the EG's nodelets. For each EC, we choose a representative block configuration x. Nodelets are found by decomposing $Q_R^{free}(x)$ into a grid of four-connected tiles, similar to the decomposition shown in Fig. 8c, and performing repeated flood fills starting at unlabeled tiles until every tile in the grid has been labeled. Each connected component is assigned an identifier that allows it to be associated with any equivalent connected components in the same EC, independent of the block configuration used to generate the tiling. Next, we need to find the directed edges of the EG, which correspond to the existence of feasible transitions between and within ECs. Feasibility calculations have already been performed in order to create the RG's edges. Since we already know the EC of each node in the RG, an edge connecting two nodes

Algorithm 4 getEquivGraph

Input: $RG = (N_r, E_r), numEC$
Output: $EG = (N_e, E_e)$
// Part 1: add node for each EC
$N_e \leftarrow \{\alpha, \beta, \gamma, \delta, \dots\}$
$E_e \leftarrow \emptyset$
// Part 2: add nodelets for each N_e
for all $\nu \in N_e$ **do**
  $\nu_i \leftarrow connectedComponents(\nu)$
  $\nu.nodelets \leftarrow \nu_i$
// Part 3: add all feasible transitions to E_e and update PG to only include feasible edges
$E_p \leftarrow \emptyset$
for all $e \in E_r \mid e = (p_1, p_2)$ **do**
  $E_e \leftarrow E_e \cup \{(p_1.ec, p_2.ec, dir(p_1, p_2))\}$
return EG

in the RG can be mapped into the EG by connecting their corresponding ECs in the EG. Self-loop edges are allowed in the EG, and they indicate that block motion can result in the block remaining in the same EC.

5.4 Constraint Generation

Recall that we have two types of constraints: manipulability and conservation of robots. Every edge in the EG has constraints of both types associated with it.

Manipulability constraints require that every nodelet, or set of nodelets, responsible for generating a transition is occupied. We say that a nodelet is responsible for a transition if a robot occupying the corresponding connected component would be able to push the block in the required direction. This leads to constraints in the form of $m_{\alpha i} \geq 1$. In the case of multiple nodelets able to execute the push, we only require that one of them be occupied, and the constraint takes the form $(m_{\alpha i} \geq 1) \vee (m_{\alpha j} \geq 1)$.

Conservation of robots constraints deal with the splitting and merging of connected components of $Q_R^{free}(x)$ when the block moves from EC α to EC β. These constraints take three different forms. In the case of merging components, we have $m_{\alpha 1} + m_{\alpha 2} = m_{\beta 1}$. For splitting components, we have $m_{\alpha 1} = m_{\beta 1} + m_{\beta 2}$. When neither merging nor splitting has occurred, we have $m_{\alpha 1} = m_{\beta 1}$.

5.5 Constraint Grouping

We have just described how constraints are assigned to edges in the equivalence graph. We now translate those constraints to the reachability graph, which is designed to propagate constraints throughout the environment in order to find a tight bound on the minimum sufficient number of robots. We begin by labeling RG edges with constraints from the EG. The construction of the graphs is such that every EG constraint maps to some RG edge, and every RG edge receives constraints from exactly one EG edge.

Algorithm 5 groupConstraints
lookupC(EG, α, β, dir) finds the constraints associated with a transition from EC α to β with the block moving in direction dir. $dir(n_1, n_2)$ for $n_1, n_2 \in N_e$ returns the direction of motion required to transition between the input nodes.

Input: $PG = (N_p, E_p), EC = (N_e, E_e)$
Output: lists of reachable constraints
// Part 1: associate constraints with each edge in the PG
for all $e \in E_p \mid e = (n_1, n_2)$ **do**
  $e.constraints \leftarrow$ lookupC(EG, n_1.ec, n_2.ec, dir(n_1, n_2))
// Part 2: all nodes receive the constraints from their departing edges
for all $n \in N_p$ **do**
  $n.constraints \leftarrow \emptyset$
  for all $e \in E_P \mid e = (n, m)$ **do**
    $n.constraints \leftarrow n.constraints \cup e.constraints$
// Part 3: backpropagate all constraints
for all $n \in N_p$ **do**
  backpropagate $n.constraints$
return maximal sets of constraints

Constraints on a directed RG edge impute constraints on the edge's parent node because they place requirements on the configuration of robots necessary to move the block as described by the edge (or some downstream edge in the graph). Constraints on a node in turn impute constraints on all inbound edges. The remaining task, then, is to fully propagate these constraint sets backward through the reachability graph until the graph becomes consistent (Algorithm 5).

In a consistent RG, every constraint on a node applies to some feasible block path starting at that node. The set of constraints in effect at any node provides an immediate solution to problem (**P2**).

To solve this problem for a general initial block configuration, we add its coordinates to the list of tile boundaries in Algorithm 2. This forces the configuration to be included as a candidate point, which results in it appearing in the RG and inheriting a list of applicable constraints.

To solve problem (**P1**), we must examine all nodes in the RG, thus representing all possible starting block configurations. The minimum sufficient number of robots will be determined by the node whose set of constraints requires the most robots.

The procedure outlined above and in Algorithm 5 is conceptually straightforward but computationally inefficient, as it can require up to $|N_r|$ traversals of the RG. The efficiency can be improved by decreasing the number of RG traversals or by reducing the size of the traversed graph. One way to do this is to condense the RG to a directed, acyclic graph whose nodes are the RG's strongly connected components. It can also help to choose an ordering for backpropagation based on heuristics.

5.6 Constrained Minimization

We have reformulated the minimum sufficient robots problem as a constrained integer minimization problem. In the worst case, solving this requires a graph search over

all possible combinations of variable assignments [12], but our problem structure will allow us to improve on this.

First, the total number of robots required is bounded above by the total number of nodelets in our EG. This is because we only require a single robot to push the block at any time, so an assignment of one robot to each connected component will be sufficient to guarantee that all manipulability constraints are satisfied.

Next, we set up the graph search. Any of the inequality constraints that are not in a disjunction imply a minimum on a single variable, so we set those variables to their minimum and propagate the constraints through the equality constraints. Once we have these minimums, we need to search over how many robots need to be added to each connected component's value in order to satisfy the constraints. This is a constraint minimization problem, so we need to guarantee that we have found the most efficient assignment, which requires searching through all possible assignments. This search can be sped up by heuristics on which variable to explore first, as well as by noting that once we have found a solution we can abandon any search path that uses more robots than the current best solution.

6 Planning

Throughout this paper we have discussed how many robots are required to enable a path for the block, but avoided discussion of any particular planning approach to generate these paths. However, the data structures we developed to solve the minimum sufficient robots problem are also useful when considering the problem of finding a feasible path for the block between a given start and goal configuration.

We present a roadmap-like planner for the world with axis-aligned rectangle obstacles. A roadmap planner uses a directed graph, where the nodes are configurations and the edges represent feasible paths between the configurations. It also requires that the graph be accessible/departible from any configuration and that it preserves connectivity [6]. We use the RG as described in Section 5.2 as the roadmap.

However, the naive approach for connecting any points $q_i, q_f \in Q_B^{free}$ to the graph does not preserve connectivity. Consider the case of a narrow hallway—if no motion perpendicular to the hallway is feasible, it is possible that there will be a feasible path between two configurations that can never reach a point in the graph. Instead, we extend the graph to include nodes corresponding to the intersection of both points' coordinates with the all other tile edges.

This operation can be performed efficiently because the RG already contains the information necessary to check the resulting edges for feasibility. Feasible directions of motion within a tile are completely determined by the feasible motions along its edges. New edges that are formed by splitting old edges inherit the same feasible direction of motion. Once the start and goal coordinates have been added to the graph, a feasible path will be found if one exists.

Now, we combine the feasible block path with information from the EG to generate constraints on where robots need to be at all times along the block path to enable block motion and ensure that there are robots positioned in the required connected components. These constraints on robot position can be fed into a standard multirobot

path planner to generate a set of robot paths that are guaranteed to push the block from start to goal.

7 Discussion and Future Work

In this paper, we have presented an algorithm that solves the minimum sufficient robots problem in any environment for which we can construct equivalence and reachability graphs. In the absence of a reachability graph, we can still compute a bound, but it is not guaranteed to be tight. We provide the required computations for the axis-aligned rectangle world. This world model is realistic for highly-structured environments, such as moving furniture through a typical office building, or pallets through a warehouse.

Additionally, the axis-aligned environment can be used to provide bounds on the minimum sufficient number of robots in more general environments. Any set of planar obstacles can be approximated to arbitrary resolution by axis-aligned rectangles. However, this approach clearly has its limits, as our algorithm's complexity scales polynomially with the number of obstacles considered. For other block shapes, we consider the bounding rectangle and largest inscribed rectangle. The bounding rectangle will give an upper bound on the number of robots needed for a given environment because it can block more possible robot paths, while the maximal inscribed square will give a lower bound on the number of robots required.

While our current work provides a complete motion planner for the block in the axis-aligned environment, it does not provide any guarantees about the optimality of the resulting path. We will investigate how to use the constraints represented in the Equivalence Graph to plan block paths that minimize the number of robots required. Similarly, we would like to be able to provide a bound on the number of robots required to push a block between any reachable pair of points in the environment. In some environments, this problem may require fewer robots than are required to push the block along any feasible path.

References

[1] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms and Implementations*. MIT Press, 2005.

[2] Mark de Berg and Dirk H.P. Gerrits. Computing push plans for disk-shaped robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010.

[3] Jonathan Fink, M. Ani Hsieh, and Vijay Kumar. Multi-robot manipulation via caging in environments with obstacles. In *IEEE International Conference on Robotics and Automation*, 2008.

[4] Russell Gayle, William Moss, Ming C. Lin, and Dinesh Manocha. Multi-robot coordination using generalized social potential fields. In *Proceedings of the IEEE international conference on Robotics and Automation*, 2009.

[5] Yoshihito Koga and Jean-Claude Latombe. On multi-arm manipulation planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1994.

[6] Steven M. LaValle. *Planning Algorithms*. Cambridge press, 2006.

[7] Kevin M. Lynch and Matthew T. Mason. Controllability of pushing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1995.

[8] Matthew T. Mason. *Mechanics of Robotic Manipulation*. MIT press, 2001.

[9] Maja J. Mataric, Martin Nilsson, and Kristian T. Simsarian. Coperative multi-robot box-pushing. In *Proceedings of IROS*, 1995.

[10] G. A. S. Pereira, V. Kumar, and M. F. M. Campos. Decentralized algorithms for multirobot manipulation via caging. *International Journal of Robotics Research*, 2004.

[11] Daniele Rus, Bruce Donald, and Jim Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995.

[12] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[13] Peng Song and Vijay Kumar. A potential field based approach to multi-robot manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

[14] Attawith Sudsang, Fred Rothganger, and Jean Ponce. Motion planning for disc-shaped robots pushing a polygonal object in the plane. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

[15] Jur van den Berg, Sachin Patil, Jason Sewall, Dinesh Manocha, and Ming Lin. Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, 2008.

[16] Jur van den Berg, Mike Stilman, James Kuffner, Ming Lin, and Dinesh Manocha. Path planning among movable obstacles: A probabilistically complete approach. In *Algorithmic Foundations of Robotics VIII*, 2008.

[17] Tsuneo Yoshikawa. Manipulability of Robotic Mechanisms. *The International Journal of Robotics Research*, 4(3), 1985.