To Push or Not to Push: On the Rearrangement of Movable Objects by a Mobile Robot

Ohad Ben-Shahar and Ehud Rivlin

***Abstract*—We formulate and address the problem of planning a pushing manipulation by a mobile robot which tries to rearrange several movable objects in its work space. We present an algorithm which, when given a set of goal configurations, plans a pushing path to the "cheapest" goal or announces that no such path exists. Our method provides *detailed* manipulation plans, including any intermediate motion of the pusher while changing contact configuration with the pushed movables.**

Given a *pushing problem*, a pushing path is found using a two-phase procedure: a context sensitive back propagation of a cost function which maps the configuration space, and a gradient descent phase which builds the pushing path. Both phases are based on a dynamic neighborhood filter which constrains each step to consider only admissible neighboring configurations. This admissibility mechanism provides a primary tool for expressing the special characteristics of the pushing manipulation. It also allows for a full integration of any geometrical constraints imposed by the pushing robot, the pushed movables and the environment.

We prove optimality and completeness of our algorithm and give some experimental results in different scenarios.

***Index Terms*—Manipulation planning, pushing planning.**

I. Introduction

PUSHING is an important, basic robotic manipulation. As with grasping, pushing is used to change both the position and the orientation of objects. However, one might claim that pushing has several advantages over grasping. It allows for easier simultaneous manipulation of groups of objects, permits the manipulation of larger and heavier objects, and most important—requires a simpler and cheaper robot structure than does grasping.

However, the action of pushing has some evident drawbacks which might make it less attractive than grasping. It is inherently restricted to a support surface (unless embedded within a very special context) which does not allow the robot to exploit the third dimension while manipulating the object. Hence, it is more likely that a specific task would lack a solution. Pushing is also mechanically unstable, and thus various control problems arise. Furthermore, pushing is different from most other motion and manipulation methods by the frequent encounters of irreversible states. A navigating robot can easily change its path upon the realization of a wrong decision. A grasping robot approaching a dead-end can usually return to a previous decision point in order to bring the object to its target. In such cases and many others, the robot can recover from wrong configurations simply by reversing its moves. This property is not one that can be used by a pushing robot due to the irreversibility of pushing. In other words, a pushing robot might push the object to a point from which no other pushing action can set it free. An example of such a hazardous configuration, which we refer to as a *trap-point*, is illustrated in Fig. 1.

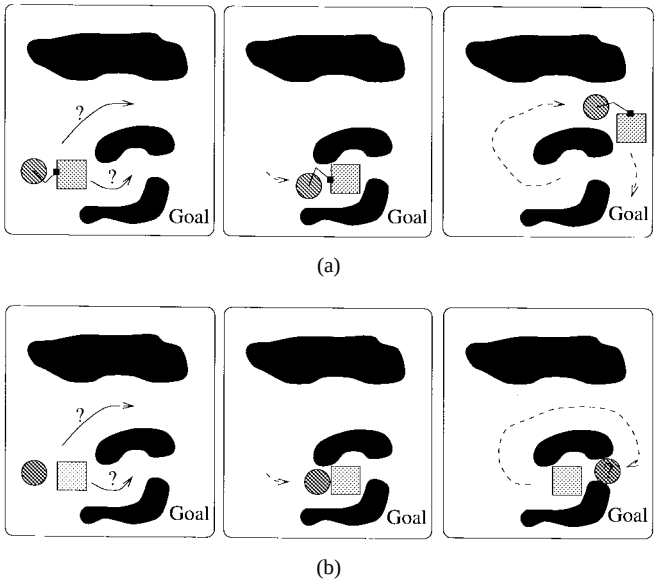


Fig. 1. The problem of trap-points: while the (a) grasping mobile robot can recover from the wrong decision by backtracking, the (b) pushing robot cannot.

Consequently, the successful completion of a pushing manipulation task requires planning. Unlike for other manipulation or navigation tasks (e.g., [14]), an online, sensory based pushing planner is likely to be either unsafe, or incomplete. We believe that high level planning tasks which involve pushing (as a special case of object manipulation) have their own characteristics and constraints, hence deserve some special attention.

While this paper deals with the issue of pushing planning, there is no doubt that using mobile robots to push objects around is an area of interdisciplinary research. In our view, a pushing problem should incorporate three main aspects in order to allow for a complete solution: the *mechanics of pushing*, *pushing planning*, and the integration of *sensing*. We refer the interested reader to [6], [15], [18], [19], [22], and [23] for key publications in the mechanics of pushing. Related work on the integration of pushing and sensing can be found in [7]–[10], [16], [21], [24], and [27].

Manuscript received September 4, 1995; revised July 5, 1997. An early version of this paper was published in *Proc. 13th Int. Conf. Robotics Automation*, Minneapolis, MN, 1996, pp. 159–164.

The authors are with the Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel.

Publisher Item Identifier S 1083-4419(98)07295-1.

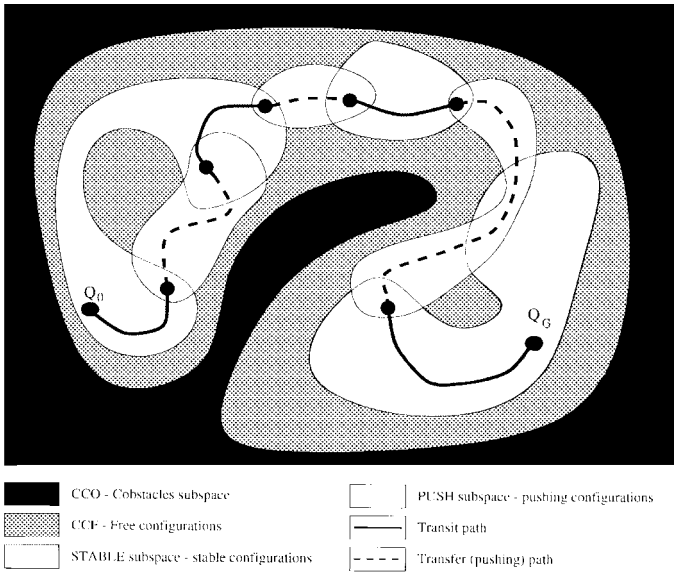


Fig. 2. The structure of a pushing C-path. Each transit path *must* reside in STABLE while each transfer/pushing path *must* reside in PUSH. Note however that transfer path must not be contained in STABLE (e.g., it is possible to push an object along a nonhorizontal surface yet it cannot be left on such a surface without the pusher's support).

Following the discussion above, we describe a basic algorithm which is capable of creating a complete pushing plan that is optimal by some cost criterion. Unlike previous work, our algorithm provides a unified approach which handles both the pushing actions, as well as any intermediate motion of the pusher while it changes contact configuration, switches from one movable to another or seeks its own goal configuration.

II. Problem Formulation and Overview

Let us consider the integrated configuration space of all nonstatic objects in the environment (a robot and movable objects). A pushing plan can be described as a special path in that space. This path, which is a constrained version of simple motion or manipulation paths, should express the constraints imposed by our manipulative action—*pushing*. The pushing problem is concerned with finding such a path, and it is discussed below.

A. The Pushing Problem

Let $\mathcal{B} = \{\mathcal{R}, \mathcal{I}, \mathcal{M}_1, \cdots, \mathcal{M}_r\}$ be a set of bodies composing the environment. $\mathcal{R}$ is a *robot* (i.e., capable of self movement), $\mathcal{I}$ represents the union of all *immovable* static bodies (i.e., obstacles), and $\{\mathcal{M}_1, \cdots, \mathcal{M}_r\}$ is a collection of *movable* rigid objects, capable of being moved by an external pushing force which might be applied by $\mathcal{R}$.

Each of the participating dynamic objects has its own configuration space. Let CS_R be the configuration space of $\mathcal{R}$ and CS_{M_i} the configuration space of $\mathcal{M}_i$. While the dimension of CS_R can be arbitrary, the dimension of CS_{M_i} is bounded by the environment in which $\mathcal{M}_i$ moves. As mentioned before, any pushing task must be carried out in a context of some support surface. We thus consider only the case of the two-dimensional environment leading to CS_{M_i} of two or three dimensions. Let $\boldsymbol{q_r}$ denote a specific configuration of $\mathcal{R}$ (i.e., a vector in CS_R) and $\boldsymbol{q}_{M_i}$ denote a specific configuration of $\mathcal{M}_i$ (i.e., a vector in CS_{M_i}).

Using the above notations, let CSS be the following Cartesian product which describes the space of common configurations of the robot and all movable objects:

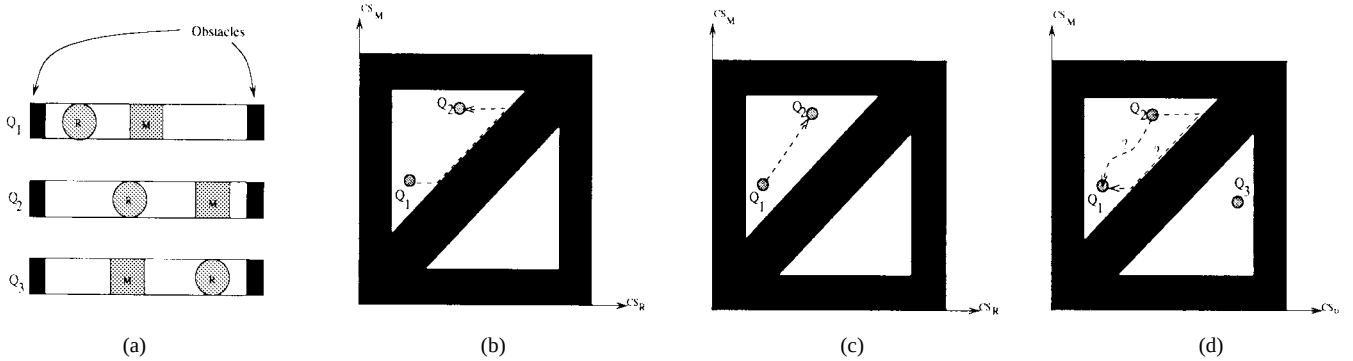
$$CSS = CS_R \times CS_{M_1} \times \cdots \times CS_{M_r}.$$

Each vector in CSS is a *composite configuration* $Q = (\boldsymbol{q}_{\mathrm{R}}, \boldsymbol{q}_{M_1}, \cdots, \boldsymbol{q}_{M_r})$ for $\boldsymbol{q_r} \in CS_R$ and $\boldsymbol{q}_{M_i} \in CS_{M_i}$. Let CCO be the C-obstacle set in CSS, i.e., the set of all composite configurations in which at least two bodies of $\mathcal{B}$ overlap (note that $CCO \neq \emptyset$ even if $\mathcal{I} = \emptyset$). Each composite configuration *not* in CCO is some legal common configuration of $\mathcal{B}$'s bodies and the set of all such configurations will be denoted by $CCF = CSS \setminus CCO$. Along with the above configuration spaces we will also use the following projection operators:

$$\Pi_R \colon CSS \mapsto CS_R$$
$$\Pi_{M_i} \colon CSS \mapsto CS_{M_i}$$
$$\Pi_R(Q) = \Pi_R(\boldsymbol{q}_R, q_{M_1}, \cdots, \boldsymbol{q}_{M_r}) = \boldsymbol{q}_R$$
$$\Pi_{M_i}(Q) = \Pi_{M_i}(\boldsymbol{q}_R, \boldsymbol{q}_{M_1}, \cdots, \boldsymbol{q}_{M_r}) = \boldsymbol{q_{m_i}}.$$

Let $N(Q)$ be the set of all neighboring configurations of Q and let $P(Q_1, Q_2)$ denote a *configuration path* (*C-path*) between Q_1 and Q_2. Similarly, let $P(Q_1, G)$ denote a C-path from Q_1 to *one of the configurations* in the set $G = \{Q_{G_1}, Q_{G_2}, \cdots, Q_{G_k}\}$. Indeed, it is clear that not every C-path, $P(Q_1, Q_2)$, is a *pushing C-path* from Q_1 to Q_2. In global terms, a pushing path is a special manipulation path and can be defined in terms coined in [2], [12], and [13]. Like any manipulation path, it is an alternate sequence of *transit paths* and *transfer paths*, as illustrated in Fig. 2. Each transit path is such that its projection on each CS_{M_i} is a single configuration (i.e., a path in which no movable object is moving). Each transfer path represents some pushing action and lies on the boundary of CCF (with at least one nonconstant projection on some CS_{M_i}). The last property follows from the fact that pushing requires physical contact in order to be realized (something which is true for most conventional manipulations, unless the force is applied remotely, e.g., via a magnetic field). Yet, since pushing is applicable by applying force only in specific directions, all transfer paths of a pushing plan are constrained further by the directions in which they can move along the boundary of CCF. This is illustrated in Fig. 3.

Equally, we can characterize a pushing path by local constraints. Let $N_A(Q)$ be the set of all *admissible* neighboring configurations of Q. Given a specific composite configuration Q, a neighboring configuration Q_1 is considered *admissible* as long as it can be reached from Q by some robot movement, with or without a pushing action involved. We will also use the complimentary definition for all configurations that a given configuration Q is their admissible neighbor. This set will be called the *admissible origin* of Q and denoted by $N_A^{-1}(Q)$. Now, a C-path $P(Q_1, Q_2)$ could be defined as a *pushing C-path* if there is a continuous parametric function



(a) (b) (c) (d)

Fig. 3. Configuration paths and pushing paths. Note that while many C-paths can be established between any two configurations, most of them cannot be considered as pushing C-paths. (a) Three placements of a robot and one movable object, both with one degree of freedom (one axis of motion). (b) The composite configuration space of the problem and a pushing C-path from Q1 to Q2. (c) Another C-path from Qi to Q2 which is not a pushing C-path. Note that this C-path includes self-movement of M. (d) No pushing path can be found from Q2 to Q1 (note that the longer path is a pulling path). Similar conclusion can be drawn regarding a pushing path from Q2 to Q3, though different reasons apply.

Θ_P: $[0,1] \mapsto CCF$ such that

$$Q_P(0) = Q_1, \quad \Theta_P(1) = Q_2$$
$$\lim_{\Delta S \to 0} \Theta_P(S + \Delta S) \in N_A(\Theta_P(S)).$$

Despite the restriction imposed by the above definition, we may still find more than one (and usually an infinite number of) pushing C-path between two given configurations. Consequently, we would like to have some measure that can be used to compare the "quality" of solutions. Let $D(Q_1, Q_2)$ be a positive cost metric in CSS. The *cost* (or *length*) of a given C-path $P(Q_1, Q_2)$ can be calculated by summing the costs between all adjacent configurations building it. More formally, under such a metric, the cost of a movement along an infinitesimal segment of the C-path is

$$dC_P = D(\Theta_P(S), \Theta_P(S + dS))$$

thus the total cost of the C-path will be

$$C_D(P) = \int_P dC_P.$$

Given a set of all C-paths between two composite configurations, we consider the minimal cost path as the *optimal* path.

Using all of the above we are ready to define our pushing planning problem.

Given $\mathcal{B}$ (a description of the environment), an initial composite configuration Q_0 and some goal composite configuration Q_G, find a pushing C-path $P(Q_0, Q_G)$, optimal by a given metric D, or report if no such path exists.

B. Solution Overview

The rest of this paper describes an algorithm which models the pushing path via local constraints and yet guarantees to produce plans which are globally legal. This approach was proved to be very easy to implement, while maintaining an impressive expressive power to deal with both the nature of the manipulation, most geometrical constraints and a large variety of artificial constraints,

Given a pushing problem, the pushing path is found using two phases: first, a *context sensitive back propagation* maps each free configuration to the cost of executing an optimal pushing plan which achieves the goal; then, this map is used to build a *specific* pushing plan from any given initial configuration to the goal configuration. Both phases evaluate the local constraints dynamically each time they need the set of admissible neighbors or admissible origins. In that sense, the neighborhood relationships in CCF are kept only implicitly. Since the cost map describes the cost of executing the pushing plan, it enables us to determine immediately whether such a plan exists or not.

In order to be numerically computable, and since exact planning is hard to be realized in a physical environment, the back propagation is carried out on a discretized version of the composite configuration space. This methodology of a potential field integrated with an approximate cell-decomposition is *resolution-complete*, so solutions are guaranteed to be found for infinitely fine resolution.

In general, the algorithm is capable of dealing with an arbitrary number of movable objects. However, the high complexity involved limits practical applications to tasks involving no more than few movable objects. It is interesting to note that although computationally extensive, the more obstacles there are, the faster the algorithm runs.

III. Relation to Previous Work

In the extensive literature that is devoted to motion-planning (see [11]), few studies address the problem of planning a pushing manipulation. Motion planning in the presence of movable objects, which covers the basic aspects of the problem, is discussed in Section III-A. The differences from current methods in assembly planning are briefly discussed in Section III-B. Some works which directly address the problem of pushing planning are covered in Section III-C.

A. Motion Planning in the Presence of Movable Objects

Having a pushing task in hand, we should instruct the robot to move in such a way that some objects in the environment

will be rearranged by the use of pushing. This variation on the basic problem, of motion planning in the presence of movable objects, was investigated in several studies, mainly in the context of grasping tasks.

Wilfong [25] was the first to analyze the complexity of the problem and showed that motion planning in the presence of movable *obstacles* is NP-hard. Moreover, in the case where the final position of the movable obstacles is specified, the problem becomes PSPACE-hard. This result convinced other researchers to concentrate first on constrained versions of the problem. Wilfong himself proposed an $O(n^3 \log^2 n)$ solution, based on an exact-cell-decomposition (see [11]), for the case of a robot translating amidst one movable polygonal obstacle and a stationary environment of complexity $O(n)$.

A generalized approach for the movable objects problem was proposed in [2], [12], and [13]. They treated the problem as a *manipulation planning* problem, applying an exact cell decomposition methodology to the *composite configuration space* of the robot and the movable objects. The planning result, namely the *manipulation path*, is an alternate sequence of *transit paths* and of *transfer paths*. Based on this scheme, a manipulation planning algorithm for a robot and one movable object amidst polygonal obstacles was presented by Dacre-Wright *et al.* [5].

While in general we expect a pushing path to have a the same structure of a manipulation path, several differences do apply. First, as mentioned in Section II, the pushing force can be applied only in specific directions (i.e., one cannot push an object by moving away from it). This observation implies that the transfer paths cannot have an *arbitrary* direction in the configuration space. Second, while Laumond *et al.* defined each transfer path to manipulate only *one* movable object, the general pushing path should allow, in our view, a simultaneous manipulation of several objects. Finally, while Laumond *et al.* defined each transfer path to represent a *rigid manipulation* (i.e., a manipulation during which the geometric relationship between the manipulator and the object remains constant), we find *nonrigid* manipulations to be more realistic, especially in the context of pushing manipulation.

B. Assembly Planning

When dealing with rearrangement problems, one may find many common aspects to *assembly planning* too. However, the state of the art research in the area of assembly planning [20] addresses problems with different characteristics than ours. Most that research ignores the manipulator, its geometry, and any constraints on the allowed manipulation (e.g., pushing only). The ultimate assembly planner should be able to generate plans directly from a CAD model of the goal assembly [26]. Consequently, assembly planning tends to ignore the initial configuration of the parts and assumes that they come from infinity. We, on the other hand, are interested in *rearrangement* of parts, i.e., changing their common configuration from a given *initial* configuration to a given *goal* configuration. In that sense, as well as in others, our rearrangement planning problem is a generalization of the assembly planning problem.

C. Pushing Planning

Despite the great deal of motion planning research, not much work has been done directly on the area of pushing planning. Akella and Mason [1] analyzed the series of pushes needed to bring a convex polygon to a desired configuration. While using pushing manipulation, this problem is a very constrained version of the rearrangement problem. They allowed only *one* convex movable object, used a simplified fence-like pusher, and ignored any other geometrical constraints (e.g., obstacles).

A comprehensive study was carried out by Lynch and Mason [17] where both mechanics, control and planning issues were considered. Their planning method was based on a *best-first* search over an inexact representation of the configuration space, which aimed at finding a path to some neighborhood of the specified goal. Again, in this work they considered only limited DOF by allowing only one movable object. It was also assumed that the fence like pusher can change the contact configuration (chosen from a discrete set) at any time, with no restrictions. As mentioned before, in this paper we are interested in multiple objects problems, where the solutions inherently integrate the motion of the pusher, including all intermediate motions between contact configurations.

Finally, a somewhat different problem was addressed by Chen and Hwang [4] who presented a practical, heuristic and inexact solution for many movable *obstacles*. Their method is primarily a motion planning method (rather than rearrangement planning) in which movable obstacles can be pushed away by the robot whenever they stand in its way to the goal.

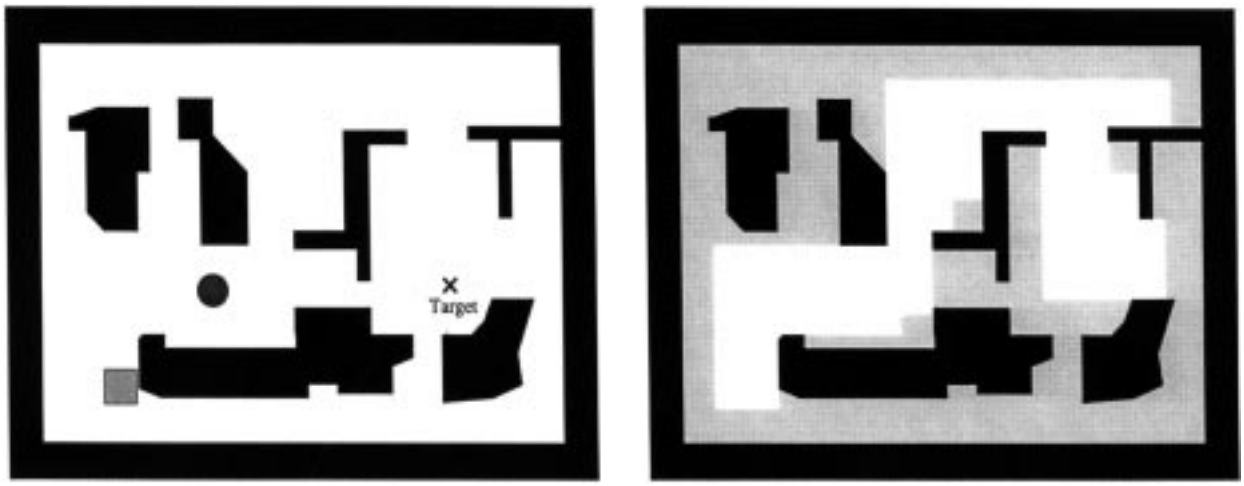
IV. Pushing Planning—A Basic Algorithm

This section describes the general pushing planning algorithm, proves its basic properties and discusses the underlying admissibility model.

A. The Algorithm

The following algorithm solves the pushing planning problem with *multiple* goal configurations. Given a *set* of goal configurations, the algorithm plans a pushing path to the "cheapest" goal, or announces that no such path exists. In addition, the algorithm is "insensitive" to the initial configuration Q_0, and is able to find an optimal pushing path from *any* initial configuration. Selecting a different initial configuration does not require major re-computation, and can be done online, as long as the set of goals remains unchanged. We should note that the cost of pushing to a specific goal depends on the initial configuration we are starting to push from. Thus, replacing the initial configuration might result in a different final goal from G.

As was mentioned before, we shall address the pushing problem under limited resolution, using a discretized version of CSS. The use of such a discretized configuration space requires no special modification in any of the above definitions. However, the term *neighboring configurations* does not refer to infinitesimal close configurations any more, and a C-path is no longer needed to be described as a continuous



(a) (b)

Fig. 4. (a) Typical pushing problem and (b) the set of its trap points. Once the object is pushed to intersect that set, no further pushing can set it free.

parametric function. Rather, any neighborhood relationship will be defined in terms of adjacent discretized configuration while C-paths will be described as a series of discretized composite configurations. Consequently, the cost/length of a C-path becomes simply the sum of the costs between all adjacent configurations that constitute it, and optimality becomes subject to the discretized version of the metric D. For simplicity, we will assume that each CSS axis is discretized with equal density (e.g., s samples per axis), although such an assumption is not required by the algorithm and might incorrectly represent the different nature of the various degrees of freedom (e.g., rotational versus translational DOF).

Given a pushing problem, an appropriate pushing path is found using a two-phase procedure. The first phase uses context sensitive back propagation of a cost function which results in a mapping of each composite configuration in CCF to the optimal cost of a pushing plan that achieves a goal configuration. The second phase restores a specific pushing C-path that needs to be executed in order to move from some initial configuration to its "nearest" goal configuration.

1) The Cost Mapping Phase: The cost mapping phase maps each point in CCF to the cost of the pushing C-path connecting it to the "cheapest" goal. This mapping is done by propagating a cost wave function originating at the target configurations. We describe this back propagation as *context sensitive* since it floods only *admissible* origins of each configuration. This phase can be considered as a preprocessing phase since it should be applied only upon a change in the set of goal configurations. In many aspects, this phase resembles the Dijkstra algorithm. However, it differs by the direction of the propagation (backward versus forward), the support of multiple goals (as opposed to Dijkstra's single source), and the fact that the graph's edges are not *a priori* known but rather constructed dynamically by the admissibility mechanism.

Let WF be the set of all CCF points currently in the wave front and let $cost(Q)$ denote the cost of a specific configuration. The following is the algorithm for building the cost map over CCF:

Initialization:
$WF \leftarrow GOAL = \{Q_{G_1}, Q_{G_2}, \cdots, Q_{G_k}\}$
$cost(Q) \leftarrow \infty, \forall Q \in CCF \setminus GOAL$
$cost(Q) \leftarrow 0, \forall Q \in GOAL$

Propagation:
while$(WF \neq \emptyset)$
Find $Q \in WF$ with minimal $cost(Q)$
$WF \leftarrow WF \setminus \{Q\}$
for each $\hat{Q} \in N_A^{-1}(Q) \cap CCF$
if $(cost(\hat{Q}) > cost(Q) + D(\hat{Q}, Q))$
then do {
$cost(\hat{Q}) \leftarrow cost(Q) + D(\hat{Q}, Q))$
$WF \leftarrow WF \cup \{\hat{Q}\}$
}
end for
end while.

To obtain a fully mapped free space, reaching Q_0 *was not* used as an additional termination condition of the propagation loop. Working with a fully mapped free space may allow quick answers for various initial configurations. It also permits the robot to regain an optimal pushing path whenever it loses its original one (due to odometry or control problems). One should note that the above-mentioned test may not reduce complexity in the worst case.

An important characteristic of the cost mapping phase is the direction in which it is carried. Clearly, the backward propagation makes the mapping insensitive to the initial configuration. Arguably, many rearrangement problems are such that their goal configuration(s) is fixed while their initial configuration may vary, hence insensitivity to the initial configuration is a desired property. However, the backward propagation serves a second, subtle point. As discussed in Section I, the irreversibility of pushing introduces trap points into the work space. While this may not be observed at the first glimpse, the set of trap points of a given task might cover major parts of the free space, as illustrated in Fig. 4. Mapping these trap points is a useless

activity, something which a planning algorithm better avoids. The backward propagation achieves exactly that. By definition, no configuration which allows pushing to the goal can have a trap-point as its admissible origin. Hence, our context sensitive back propagation can never reach a trap-point. This is not true for a forward propagation, though.

2) The Pushing C-Path Restoration Phase: After the whole space is mapped, a pushing C-path can be built from any initial configuration simply by using a variation of a gradient following hill-climbing. This process should preserve the notable difference between a general C-path and a pushing C-path, hence only *admissible* neighboring configurations are considered in each step. The same constraint was used in the propagation phase by limiting the propagation to $N_A^{-1}(Q)$ alone. Given an initial configuration Q_0, the following procedure restores the appropriate pushing C-path or reports whether no such path exists. We will use the $\bowtie$ symbol to denote a concatenation of a new point to the path and the $select(\ )$ operator for arbitrary selection of an element from a set.

Restoration:
if ($Q_0 \notin CCF$ or $cost(Q_0) = \infty$)
NO PUSHING PATH EXISTS !!!
else
$P(Q_0, GOAL) \leftarrow Q_0$
$Q_C \leftarrow Q_0$
while $Q_C \notin GOAL$
$Q_C \leftarrow select\{Q | Q \in N_A(Q_C) \wedge$
$cost(Q) = cost(Q_C) - D(Q_C, Q)\}$
$P(Q_0, GOAL) \leftarrow P(Q_0, GOAL) \bowtie Q_C$
end while
OUTPUT $P(Q_0, GOAL)$.

The core of this procedure is the selection of the next configuration. The $select(\ )$ operator is used since the cost map might contain saddles and the optimal pushing C-path might not be unique (this is easily seen for cases of symmetrical solutions). The cost equivalence test is needed to ensure that we select only optimal moves. Choosing, instead, the minimal cost admissible neighbor as the next configuration in the C-path might lead to nonoptimal pushing C-paths since it practically ignores the cost metric. Integrating the metric test is equivalent to keeping a pointer to the parent node, as done in many graph search algorithms. Fig. 5 demonstrates such a case where following minimal admissible neighbors results in a nonoptimal pushing C-path.

B. The Role of Admissibility

A crucial part in the algorithm is the involvement of admissible neighbors and origins. Basically, those are required in order to produce legal C-paths (i.e., *pushing* C-paths) and avoid trap-points, as discussed in Section IV-A1. However, an appropriate definition of admissible neighbors/origins of a configuration can also be used to control the specific pushing functionality of the robot, and to present other behaviors

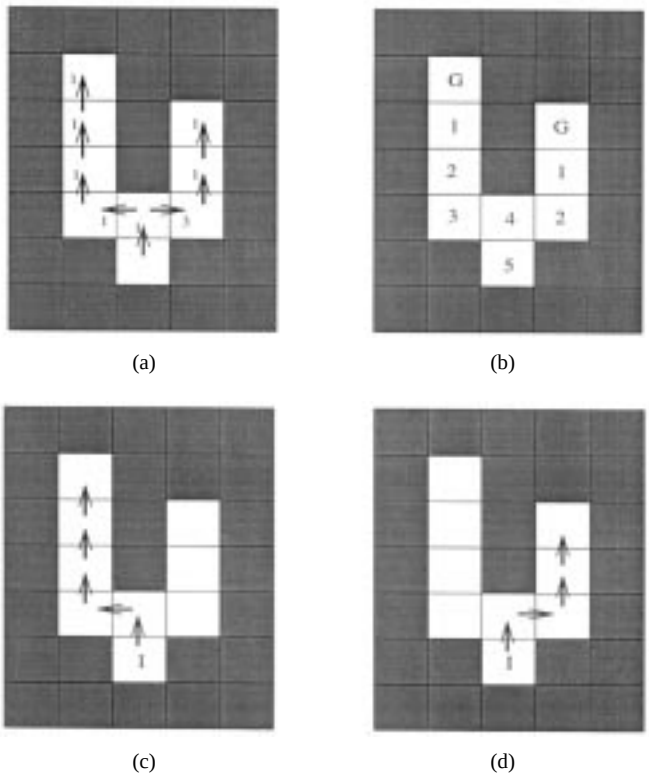


Fig. 5. The role of the metric during the restoration phase. (a) CCF, admissible neighbors and metric. (b) Result of the cost mapping phase. (c) Output of the restoration phase. (d) Path found by minimal cost neighbor test.

into the planning. Notable examples of admissibility-based controllable features are

- allowable pushing directions;
- allowable contact points which may serve for pushing (note that after selecting the allowable pushing directions, the contact points are space variant in the general case since the friction distribution is not necessarily uniform);
- the number of objects that can be pushed at once and their relative positions while being pushed together.

Any behavior chosen for the pushing planning affects the size of $N_A(Q)$ and $N_A^{-1}(Q)$ and the complexity of their computation. In most of the cases, we can calculate those sets using only local information (i.e., based on the "coordinates" of the configuration alone), yet an accurate calculation requires a full model of the pushing mechanics (including friction distribution, movable object parameters, etc.), which we assume to be given by some external source. In order to clarify this point, let us examine a task of pushing one movable object when the only allowable manipulation is a pure translational pushing to the left. Let us assume that the COF (center of friction) of the object is known and that the robot can push the movable object to the left only after maintaining a relative position of $(\Delta x, \Delta y)$, as illustrated in Fig. 6.

Given a configuration Q, $N_A^{-1}(Q)$ can be obtained by using the following two steps.

1) Consider all neighboring configurations from which Q could be reached by independent movement of $\mathcal{R}$. In

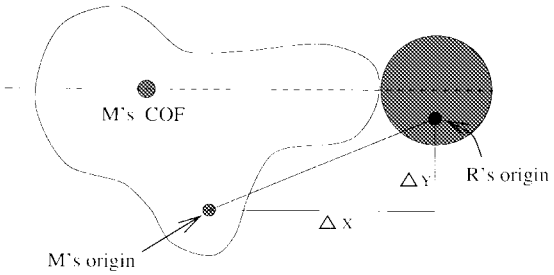


Fig. 6. The relative position that is needed to allow a translational pushing to the left.

other words, all configurations $\hat{Q}$ such that

$$\hat{Q} \in CCF \cap N(Q) \ \wedge \ \Pi_M(\hat{Q}) = \Pi_M(Q).$$

2) Consider all configurations from which $\mathcal{R}$ could have pushed $\mathcal{M}$ toward Q. In our case, if

$$\Pi_R(Q) - \Pi_M(Q) = (\Delta x, \Delta y)$$

then $\mathcal{R}$ and $\mathcal{M}$ are in accurate contact mode for the allowable pushing. Thus, the configuration $\hat{Q}$ from which $\mathcal{R}$ could have pushed $\mathcal{M}$ toward Q should also be considered as an admissible origin. Calculating $\hat{Q}$ is easy since it can be directly obtained by subtracting the left pushing vector $(-\delta, 0, -\delta, 0)$, from Q.

While the admissibility mechanism serves as a primary tool for expressing pushing abilities, it can be used to describe environmental-based behaviors as well. We can use it to

- adjust pushing abilities (e.g., contact points and pushing directions) in accordance with environmental parameters (e.g., the normal to the support surface);
- limit the areas in which the robot can dismiss a movable object after pushing it (e.g., no object should be dismissed on an oblique area);
- force the robot to keep a minimal distance from an object when not pushing it;
- prevent the robot from roaming or pushing in a nonsafe distance from obstacles;
- allow self movement of objects in some areas due to environmental influence (such as gravity).

Some of these behaviors are simulated in Section V.

C. Cost Metrics and Usage

While the admissibility mechanism allows us to distinguish between legal manipulations and illegal ones, it is not sufficient for weighting each manipulative action. A notable example is the different weights which are likely to be allocated for the action of pushing an object and the action of pure movement when no pushing is involved, something which is not applicable via admissibility. Thus, while the admissibility mechanism limits the planning to *legal* solutions, the cost metric enables the preference of one legal solution over another.

An intuitive usage of the metric could be weighting the pushing actions (e.g., by using the total mass of the objects currently being pushed). Additional possibilities could involve the current region's passability (i.e., how hard is it to move in a region), the distance from obstacles (in order to achieve safer plans when possible), changes in object-robot contact modes (in order to minimize them), etc. Some of these features are illustrated in the simulations of Section V.

D. Correctness

In order to show that the algorithm is correct, we should prove the following properties.

Claim 1: Given a pushing problem, the propagation phase is guaranteed to stop.

Proof: Each iteration of the propagation is guaranteed to produce exactly one new configuration, the cost of which will never be overwritten. This property holds for the minimal cost configuration extracted from the wave front at the beginning of each iteration. Since the number of available configurations in a discretized CCF is bounded, the propagation is guaranteed to stop. □

Claim 2: Given a cost map (produced by the propagation phase) and an initial configuration, the restoration phase is guaranteed to reach (one of) the goal configuration(s).

Proof: Since the restoration phase is a hill-climbing-like search, we should verify that local minima are not likely to occur. Let us assume that such an event has occurred, i.e., that the propagation phase attached a cost $c \neq 0$ to a configuration $Q \notin GOAL$, while $cost(\hat{Q}) > c, \forall \hat{Q} \in N_A(Q)$. If that has happened, no pushing C-path which tends to pass through Q will reach the goal. Following the definition of $N_A(Q)$, it is clear that $Q \in N_A^{-1}(\hat{Q}), \forall \hat{Q} \in N_A(Q)$, i.e., Q is an admissible origin of each of its admissible neighbors, and **only** of them. Although the propagation phase assigns higher cost to the origin of a configuration, this cost may be later overwritten by a lower cost. Still, even if such an overwrite occurs, it must originate from another admissible neighbor of Q since the propagation is done locally. Thus, we can say that $\exists Q_1 \in N_A(Q)$, such that $cost(Q_1) = cost(Q) - D(Q, Q_1) < cost(Q)$. This fact contradicts our assumption and proves the claim. We should note that the only configurations that the above claims do not stand for are the goal configurations where indeed the algorithm should stop. This means that the stopping condition of the restoration phase could ignore the $GOAL$ set and the restoration could continue until it encounters a cost minimum. □

Claim 3: The C-paths produced by the algorithm are indeed pushing C-paths.

Proof: The restoration phase considers only admissible neighbors while constructing the C-path. Hence, by construction, this property is maintained. □

Claim 4: The pushing C-paths produced by the algorithm are optimal.

Proof: Let us assume the algorithm restored the C-path $P_1(Q_0, GOAL) = (Q_0, Q_1, \cdots Q_m)$ while the optimal, less expensive C-path from Q_0 to $GOAL$ is $P_{\text{optimal}}(Q_0, GOAL) = (Q_0, Q_1^O, \cdots, Q_n^O)$. Note that the final configurations Q_m and Q_n^O need not be the same, yet both belong to $GOAL$. Let the i's configurations in those C-paths be the first in which they vary, i.e., $Q_i \neq Q_i^O$ while $Q_j = Q_j^O, \forall j < i$ (note that $i > 0$ since at least Q_0 must share the C-paths). Fig. 7 illustrates this situation where P_1 and P_{optimal} depart.

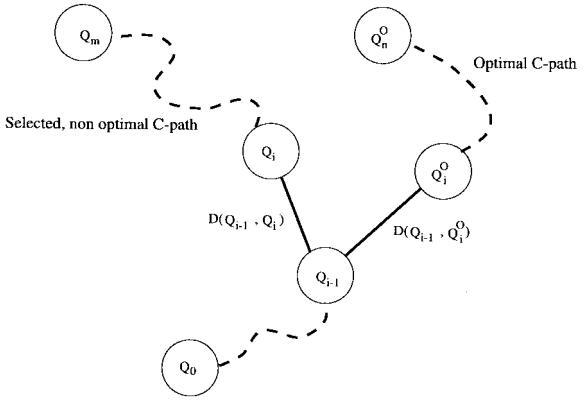


Fig. 7. The split point of the nonoptimal C-path from the optimal C-path.

Let us compare the costs of the configurations at the split point. Following the propagation phase and the optimality of P_{opt}, we conclude that the cost of Q_{i-1} should be

$$cost(Q_{i-1}) = \min(C_{\text{select}}, C_{\text{opt}}) = C_{\text{opt}}$$

for

$$C_{\text{select}} = cost(Q_i) + D(Q_{i-1}, Q_i)$$
$$C_{\text{opt}} = costQ_i^O + D(Q_{i-1}, Q_i^O).$$

Following the restoration phase and the production of P_1, we conclude that the cost of Q_i should be

$$cost(Q_i) = cost(Q_{i-1}) - D(Q_{i-1}, Q_i).$$

Integrating the above two equations allows us to conclude that our assumption is admissible under the following condition:

$$cost(Q_i) + D(Q_{i-1}, Q_i) = cost(Q_i^O) + D(Q_{i-1}, Q_i^O)$$

which holds when either $Q_i = Q_i^O$ or $C_D(P_1) = C_D(P_{\text{opt}})$. Since both conditions contradict the assumption, our claim has been proved. □

E. Complexity

It has already been proved that our pushing problem is PSPACE-hard [25], which implies a complexity exponential in CSS's dimension. Using an equal density discretization of s samples per CSS axis, and assuming n movable objects pushed by a 3-DOF robot in a 2-D environment, we get a worst case complexity of $O(V \cdot (f(n) + \log(V)))$ with $V = s^{3+3n}$. The function $f(n)$ represents the fact that the set of admissible neighbors of a configuration might affect the complexity. Choosing a simple pushing behavior, whereby the robot can push only one object at a time, will limit $f(n)$ to $O(n)$. Allowing the pushing of up to two objects at once will bound $f(n)$ by $O(n^2)$, and allowing any combination of the n objects to be pushed by the same robot action will make $f(n)$ exponential.

As mentioned before, the greater the number/area of the obstacles, the less expensive is the algorithm. As this happens, CCF gets smaller, and thus the maximum number of propagation iterations is decreased. In any case, the complexity of the current version of the algorithm makes it practical for simple scenarios only, allowing probably no more than few movables to be involved in planning under a rough discretization resolution.

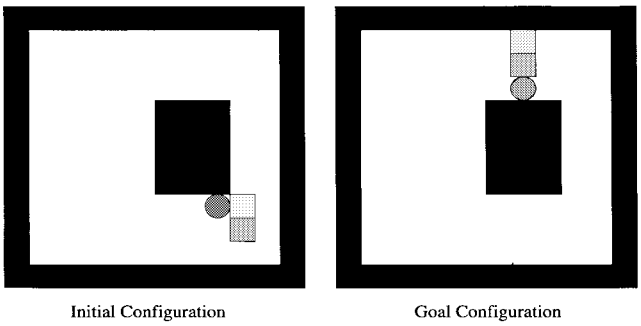


Fig. 8. Test case A.

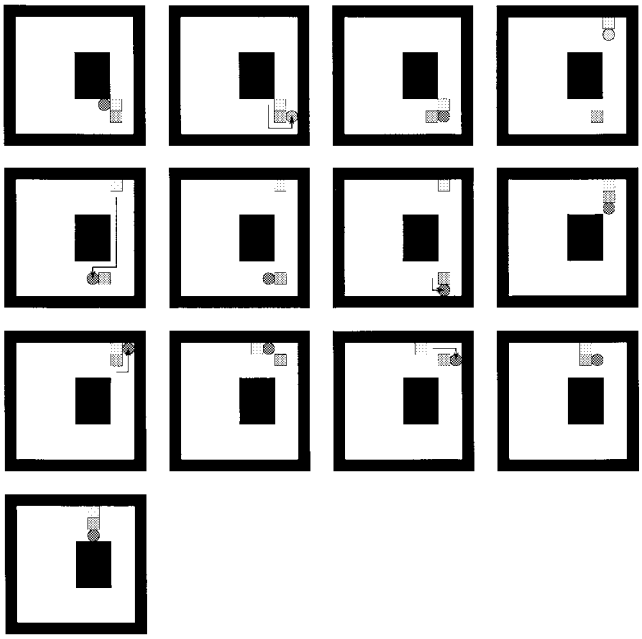


Fig. 9. Pushing plan A1—the robot was allowed to push only one box at a time. No constraint was imposed on the distance that $\mathcal{R}$ can get from the boxes. The only invoked metric was the *mass metric*.

V. Experimental Results

A. Simulated Examples

In order to examine the algorithm, it was implemented and tested in a simulated environment. Demonstrated below are the results of some test cases which involve pushing tasks with a 2-DOF robot. In order to present the various abilities of the admissibility mechanism, as well as different features of the metric function, the simulated environment was designed to support the following parameters. Note that the first three behaviors are "admissibility-based" (i.e., implemented by the admissibility mechanism), while the others are "metric-based."

- The maximum number of objects that could be pushed simultaneously was user defined. Our simulation supported

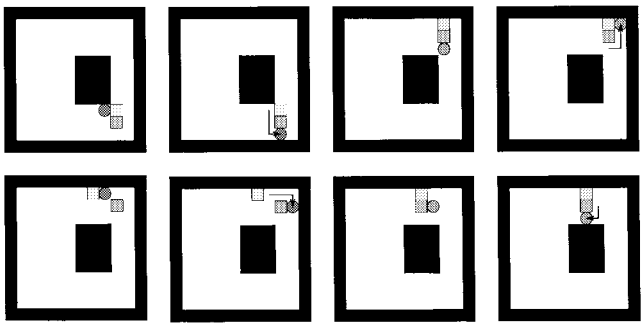


Fig. 10. Pushing plan A2, in which two boxes could be pushed simultaneously. Other parameters were similar to those of plan A1.

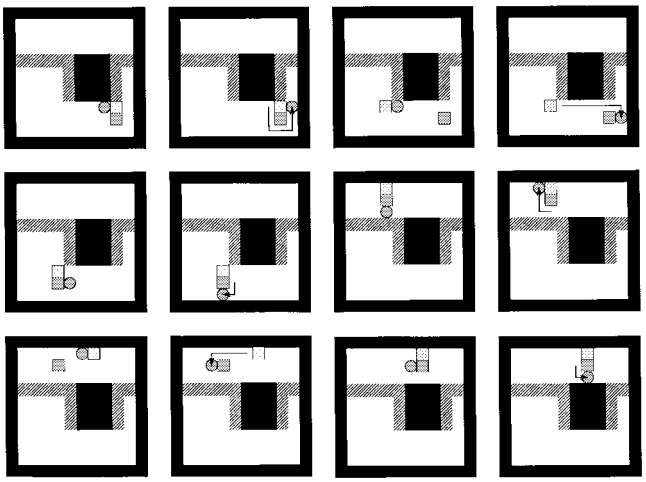


Fig. 11. Pushing plan A3—the robot was allowed to push up to two boxes and the *passability metric* was integrated with the *mass metric*. Note that this time, the A1 solution corresponds to a long travel through the "rough" area (marked), hence it becomes more expensive than the presented optimal path.

up two movable objects, which means that this limit could be set to 1 or 2.

- Objects were allowed to have 2- (only translation) or 3- (translation and rotation) DOF. Translations were limited to the main directions (north, south, west, and east) and rotations were limited to 90° (see discussion below). In addition, each movable object was associated with a user defined set of pushing contact modes. This set was then used by the robot when pushing the object.
- A limit could be set on the (Manhattan) distance that the robot could keep away from each of the movable objects (while not pushing it).
- The planner weighted each movement by the total moved mass (i.e., the more objects are pushed, the more expensive is the movement). We denote this basic metric as the *mass metric*.
- The planer could be set to prefer "safe" pushing paths (i.e., paths in which the movable objects tend to keep their distance from the obstacles). We denote this metric as the *safety metric*.
- The planner could be set to prefer pushing paths that go through regions with better "passability." The simulator allows up to three different levels of passability and whenever this feature was invoked, the "rough" areas were marked accordingly. We denote this metric as the *passability metric*.

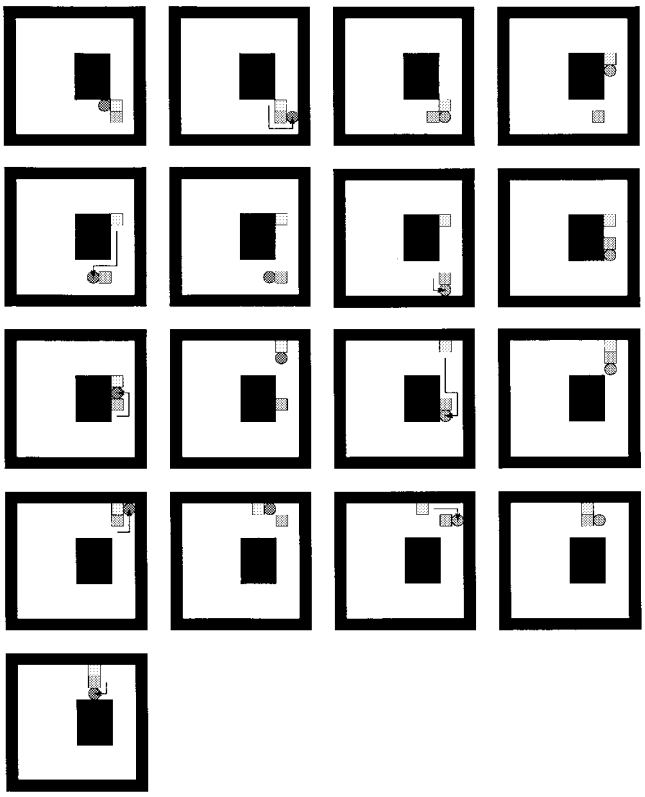


Fig. 12. Pushing plan A4—pushing was limited to one box at a time. The (Manhattan) distance the robot could keep from the box which was not being pushed was limited to 7 units. The only metric used was the *mass metric*. Note how the robot obeys the distance constraint by leaving the first box in order to bring the second one closer.

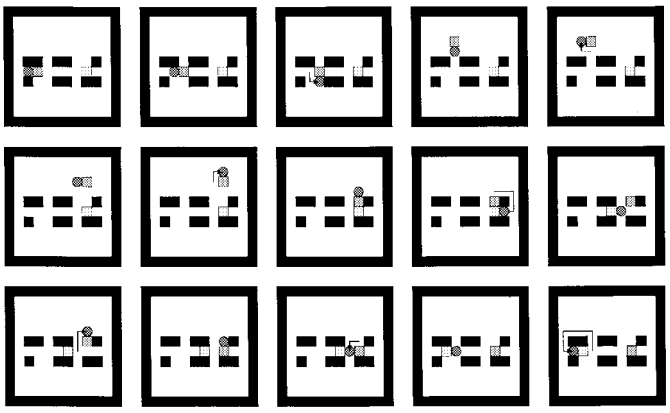


Fig. 13. Pushing plan B1—the robot should exchange the boxes, thus it must push at least one box to an intermediate position in order to clear its way.

The first test case deals with the pushing task presented in Fig. 8. Running the algorithm produced the results illustrated in Figs. 9–12, showing the influence of the above parameters on the planning. Planning steps are ordered left to right, top to bottom. Transit paths are marked by arrows. Note that while admissibility-based features are guaranteed to be

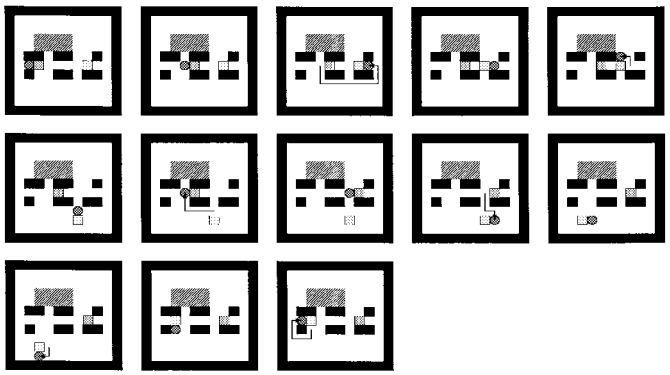
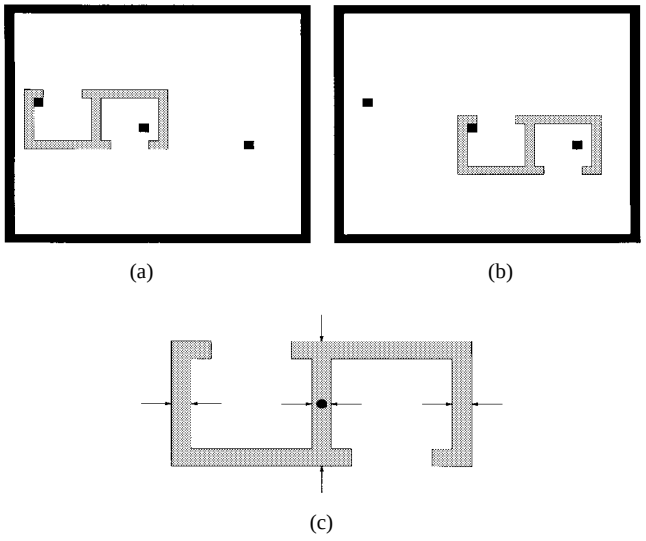


Fig. 14. Pushing plan B2—the marked region was defined as "rough" and the *passability metric* was integrated into the planning. Note that this time the optimal solution is totally different than plan B1, and the "rough" area is avoided.



(a) (b) (c)

Fig. 15. Test case C—pushing task with a complex movable. (a) and (b) show the goal and initial states, respectively, while (c) details the movable object and all translational pushing points which correspond to the marked COF.

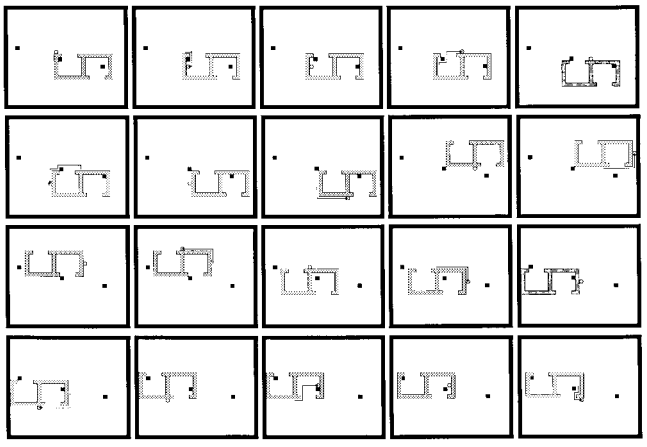


Fig. 16. Pushing plan C1—pushing is done with a small robot.

realized, metric-based features are considered by the planner as "recommendations."

The second test case demonstrates a known feature of manipulation tasks: the need to handle conflicts by designing

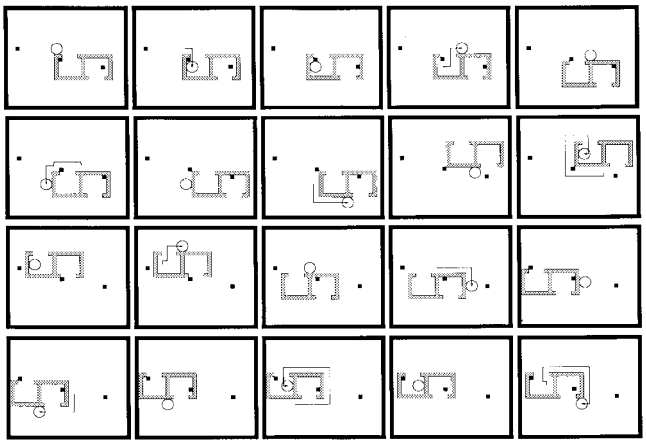
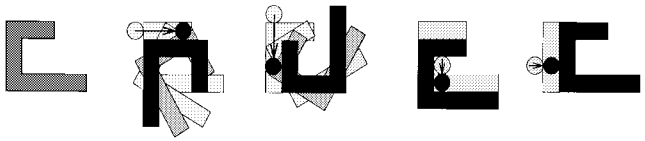


Fig. 17. Pushing plan C2—pushing is done with a larger robot. Note that some moves carried out by the smaller robot are not feasible any longer. Thus, an alternative (and more expensive) action is chosen.



(a) (b) (c) (d) (e)

Fig. 18. Test case D—planning with nontranslational pushings (i.e., pushings which not only translate the object but also rotate it). (a) details the movable object while (b)–(e) show all the allowed pushings (relative contact point and the pushing outcome).

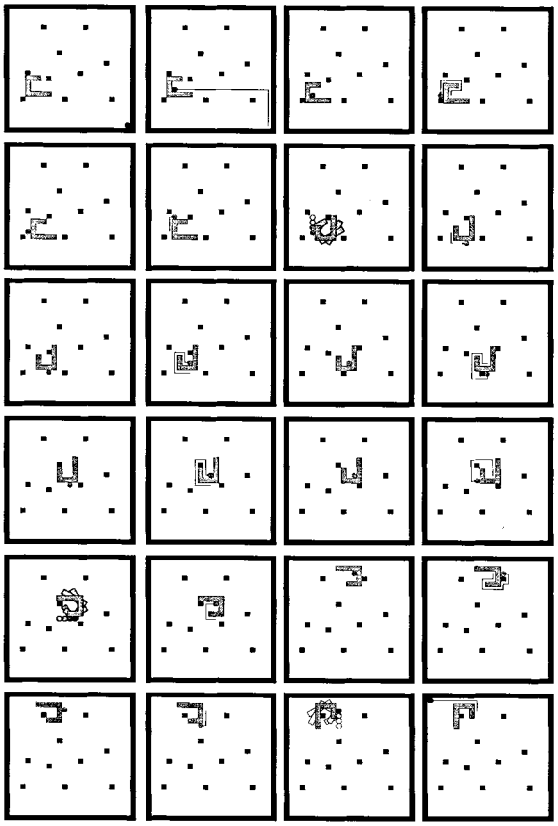


Fig. 19. Pushing plan D1—using nontranslational pushings in a cluttered environment.

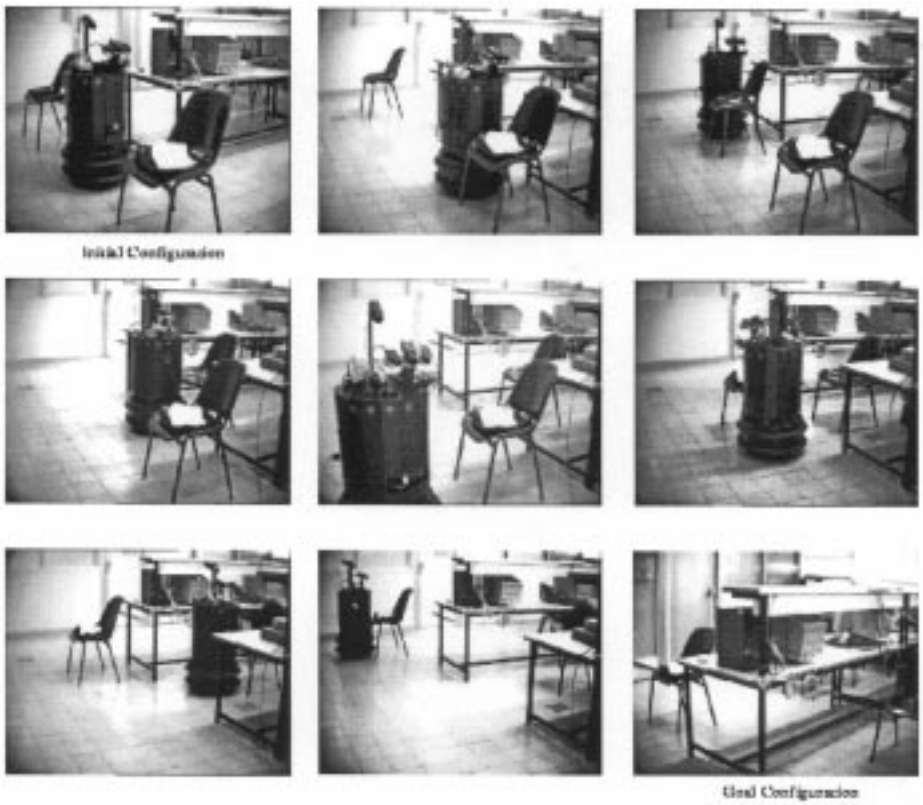


Fig. 20. Real-world experiment snapshots.

and solving subgoals. This feature is managed automatically by our planner and is illustrated in Figs. 13 and 14.

The next test case deals with pushing plans for complex movable object and the impact of the pusher's geometry on the produced plan. Fig. 15 sketches the shape of a complex movable object, the task to be solved, and all allowed pushings. Figs. 16 and 17 illustrate the planning results for a small and a large pusher, respectively. Note that some moves carried out by the smaller robot could not be executed by the larger one, thus an alternative action was chosen.

Although the algorithm is general, its discretized implementation imposes severe limitations with regard to nontranslational pushings (i.e., pushings which not only translate the object but also rotate it). Supporting these kinds of pushings requires either an infinite discretization resolution or the integration of some sub-cell manipulation mechanism into the algorithm. Nevertheless, a special case of nontranslational pushings—those that rotate the object by 90° and align it back with the discretization grid—can be supported. Assume that a mechanical analysis of the object in Fig. 18(a), results in the demonstrated pushing manipulations. Fig. 19 shows a pushing plan in a cluttered environment. Avoiding rotational pushings in that environment would prevent us from finding a solution to most rearrangement problems, including the one presented in the figure.

B. Real-World Experiment

The experiments so far were executed in a simulated environment. However, as our final goal is a real working system, we tested our algorithm in a realistic scenario using a mobile platform. The planner was run on a Sun 4/460 computer where it was integrated with the control environment of the NOMAD-200 mobile robot from Nomadic Inc. The planner used a coarse representation of the lab where the test was carried out, and each run involved two movable objects. Only two types of movables were used in this experiment: boxes and chairs. Fig. 20 shows some snapshots from one execution while Fig. 21 details the corresponding plan.

Arguably, as the research outlined in this paper deals with planning issues, the described experiment may not necessarily contribute significant information. However, we found it important to experiment with a real platform for two reasons. First, we felt that such an experiment may reveal additional features to be integrated in future planning algorithm. For example, while developing our simulated environment, we assumed a holonomic pusher which draws no costs for changing direction of motion. The Nomad 200, as a sync drive platform, was found to require quite a long time for each turn, though. This immediately suggested an additional metric to be integrated in the planning. Second, we expected such an experiment to show how really critical are the other aspects of the pushing rearrangement problem, namely the need for sensory feedback and the integration of accurate mechanical model. Not unexpectedly, the result emphasized the great importance of sensory feedback. Before each run the robot and the movables needed to be carefully calibrated for position and orientation. Although small odometry errors are acceptable in most cases for indoor navigation, this is not the case for pushing. Missing, even slightly, the correct pushing contact

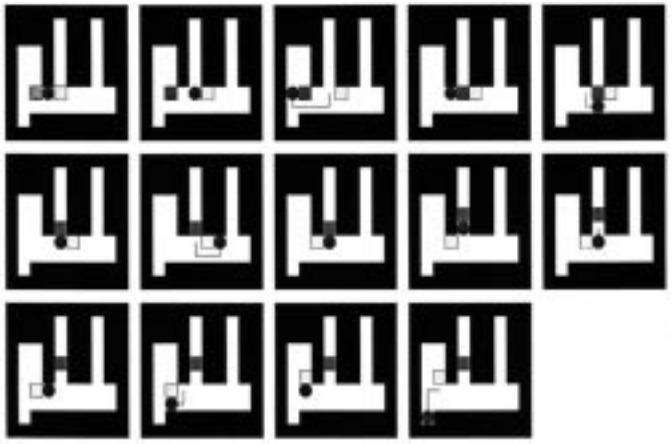
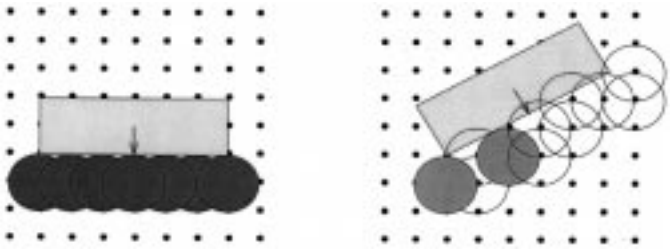


Fig. 21. Pushing plan for the real-world experiment.



(a) (b)

Fig. 22. The problem of rotational pushing. (a) Describes a square object aligned to the grid. The dots show all possible (discretized) configurations that the robot can take, and the circles illustrate all the configurations on the grid that define a contact mode of the robot and the labeled edge. (b) Shows the same object, oriented at some angle. Note that in this case only two configurations of the robot still define some contact mode with regard to the same edge.

mode can be critical. The two different movables we used showed different behaviors when pushed and demonstrated the importance of the integration of a sensory information and a mechanical model of the manipulation.

VI. Discussion

We formulated the problem of planning a pushing manipulation by a mobile robot which tries to rearrange several movable objects in its work space. Our resolution-complete algorithm uses a two-phase procedure: first, it propagates a cost function to achieve a full mapping of every free configuration, and then it restores a specific pushing C-path using a hill-climbing search from the given initial configuration to the global. By restricting both the propagation and the restoration to admissible origins/neighbors alone, only legal pushing paths were produced by the algorithm. The admissibility mechanism provides a primary tool for expressing the special characteristics of the pushing manipulation. It also allows a full integration of any geometrical constraints imposed by the pushing robot, the pushed objects and the environment. The fact that the propagation is carried out backward also allows it to avoid mapping the trap points, a property that may be significant in certain environments.

Our algorithm has been shown to be optimal and (resolution-) complete. As for other approximate cell decomposition algorithms, the greater the number/area of obstacles, the less expensive is the algorithm. As this happens, CCF gets smaller, and the maximum number of propagation iterations is decreased. Although one can use the planning algorithm in practice for scenarios with few movables under a relatively sparse resolution, its complexity makes it hard to use for most real-life applications. In another paper [3], we begin to study planning methods which compromise between the complexity of the computation and the completeness (or optimality) of the planning.

Before implementing the proposed algorithm on a real pushing robot, one needs to consider several issues and problems. The first relates to the need for integrating arbitrary rotational pushing into the planner. This kind of pushing manipulations, which are common in everyday pushing actions, cannot be easily supported by our planner due to the discretization of the configuration space. Naturally, in such a discretized space, both the position and the orientation of each body are allowed to get only discretized values. Hence, it is likely that the object can reach some orientation in which the robot will not be able to achieve all the allowed contact modes needed for the pushing (see Fig. 22). Consequently a naive implementation of rotational pushings in the framework of our algorithm may cause the planner to miss pushing paths or create illegal ones. As demonstrated in Section V, rotations which align the object back to the grid and allow the planner to correctly consider all contact modes, can still be implemented without any significant change.

Another issue that cannot be avoided is the need to handle scenarios where the planner has incomplete knowledge. In most rearrangement applications we can expect that at least the initial configuration of the movable objects will not be known *a priori*. Hence, an appropriate exploration strategy that allows the robot to learn where each movable lies must be carried out before the pushing plan is created. However, when some movables cannot be localized, the robot might need to push objects which block critical passages. Such an action should be taken with caution, carried out as a mini-plan, since a precipitious move might push an object to a trap-point and prevent any chance of succeeding with the mission.

The last issue that needs to be addressed is the pushing itself. As the action of pushing is inherently unstable (especially for point pushers), we might need an adaptive strategy which exploits necessary sensory information in order to allow controllable pushing actions. Sensory based pushing, as well as the other issues mentioned above, are topics for our future research.

References

[1] S. Akella and M. Mason, "Posing polygonal objects in the plane by pushing," in *IEEE Int. Conf. Robotics Automation*, May 1992, pp. 2255–2262.

[2] R. Alami, T. Simeon, and J. Laumond, "A geometrical approach to planing manipulation tasks. the case of discrete placements and grasps," in *Int. Symp. Robot. Res.*, 1989, pp. 453–463.

[3] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Trans. Robot. Automat.*, vol. 14, pp. 549–565, Aug. 1998.

[4] P. Chen and Y. Hwang, "Practical path planning among movable obstacles," in *IEEE Int. Conf. Robotics Automation*, 1991, pp. 444–449.

[5] B. Dacre-Wright, J. Laumond, and R. Alami, "Motion planning for a robot and a movable object amidst polygonal obstacles," in *IEEE Int. Conf. Robotics Automation*, May 1992, pp. 2474–2480.
[6] M. Erdmann and M. Mason, "An exploration of sensorless manipulation," *IEEE J. Robot. Automat.*, vol. 4, pp. 369–379, Aug. 1988.
[7] P. Franchi, F. Gandolfo, P. Casalino, G. Sandini, and R. Zaccaria, "Preliminary experiments of visuo-motor integration in pushing tasks," in *IEEE Int. Workshop Intelligent Robots Systems (IROS)*, Nov. 1991, vol. 2, pp. 535–537.
[8] F. Gandolfo, M. Tistarelli, and G. Sandini, "Visual monitoring of robot actions," in *IEEE Int. Workshop Intelligent Robots Systems (IROS)*, Nov. 1991, pp. 269–275.
[9] H. Hügli, C. Facchinetti, and F. Tiéche, "Multi-layered hybrid architecture to solve complex tasks of an autonomous mobile robot," in *Proc. 3rd GWIC Intelligent Systems*, 1994.
[10] M. Inaba and H. Inoue, "Vision-based robot programming," in *Int. Symp. Robotics Research*, 1990, pp. 129–134.
[11] J. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.
[12] J. Laumond and R. Alami, "A new geometrical approach to planning manipulation tasks. The case of a circular robot and a movable circular object amidst polygonal obstacles," Tech. Rep. 88314, LAAS, Toulouse, France, 1988.
[13] ———, "A geometrical approach to planing manipulation tasks in robotics," Tech. Rep. 89261, LAAS, Toulouse, France, 1989.
[14] V. Lumelsky and A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, no. 4, pp. 403–430, 1987.
[15] K. Lynch, "The mechanics of fine manipulation by pushing," in *IEEE Int. Conf. Robotics Automation*, May 1992, pp. 2269–2776.
[16] K. Lynch, "Estimating the friction parameters of pushed objects," in *IEEE Int. Workshop Intelligent Robots Systems (IROS)*, July 1993, pp. 186–193.
[17] K. Lynch and M. Mason, "Stable pushing: Mechanics, controllability, and planning," in *1st Workshop Algorithmic Foundation Robotics*, 1995.
[18] M. Mason, "Mechanics and planning of manipulator pushing operations," *Int. J. Robot. Res.*, vol. 5, pp. 53–71, Fall 1986.
[19] M. Mayeda and Y. Wakatsuki, "Strategies for pushing a 3-D block along a wall," in *IEEE Int. Workshop Intelligent Robots Systems (IROS)*, Nov. 1991, pp. 461–466.
[20] *Computer-Aided Mechanical Assembly Planning*, L. Momem de Mello and S. Lee, Eds. Norwell, MA: Kluwer, 1991.
[21] Y. Okawa and K. Yokoyama, "Control of a mobile robot for the push-a-box operation," in *IEEE Int. Conf. Robotics Automation*, May 1992, pp. 761–766.
[22] M. Peshkin and A. Sanderson, "The motion of a pushed, sliding workpiece," *IEEE J. Robot. Automat.*, vol. 4, pp. 569–598, Dec. 1988.
[23] D. Pham, K. Cheung, and S. Yeo, "Initial motion of a rectangular object being pushed or pulled," in *IEEE Int. Conf. Robotics Automation*, 1990, pp. 1046–1050.
[24] F. Tiéche, C. Facchinetti, and H. Hugli, "Architecture of an autonomous system: Application to mobile robot navigation," in *Proc. Symp. Artificial Intelligence Robotics*, Oct. 1994.
[25] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proc. ACM Symp. Computational Geometry*, 1988, pp. 279–288.
[26] J. Wolter, "On automatic generation of assembly plans," in *Computer-Aided Mechanical Assembly Planning*, L. Homem de Mello and S. Lee, Eds. Norwell, MA: Kluwer, 1991, ch. 11, pp. 263–288.
[27] T. Yoshikawa and M. Kurisu, "Identification of the center of friction from pushing an object by a mobile robot," in *IEEE Int. Workshop Intelligent Robots Systems (IROS)*, Nov. 1991, vol. 2, pp. 449–454.

Ohad Ben-Shahar received the B.Sc. and M.Sc. degrees in computer science from the Technion, Israel Institute of Technology, Haifa, in 1989 and 1996, respectively. He is currently pursuing the Ph.D. degree in computer science at Yale University, New Haven, CT.

His research interests are computer vision and robot motion/manipulation planning.

Ehud Rivlin received the B.Sc. and M.Sc. degrees in computer science and the M.B.A. degree from the Hebrew University, Jerusalem, Israel, and the Ph.D. degree from the University of Maryland, College Park.

He is currently an Assistant Professor in the Computer Science Department at the Technion, Israel Institute of Technology, Haifa. His current research interests are in machine vision and robot navigation.