To appear in *Advanced Robotics*
Vol. 00, No. 00, September 2017, 1–16

FULL PAPER

Tradeoffs in the Computation of Minimum Constraint Removal Paths for Manipulation Planning

Athanasios Krontiris[a] and Kostas E. Bekris[a*]

[a]*Rutgers, The state University of New Jersey, New Jersey, USA*

(*September 2017*)

The typical objective in path planning is to find the shortest feasible path. Many times, however, such paths may not be available given constraints, such as movable obstacles. This frequently happens in manipulation planning, where it may be desirable to identify the minimum set of movable obstacles to be cleared to manipulate a target object. This is a similar objective to that of the Minimum Constraint Removal (MCR) problem, which, however, does not exhibit dynamic programming properties, i.e., subsets of optimum solutions are not necessarily optimal. Thus, searching for MCR paths is computationally expensive. Motivated by this challenge and related work, this paper investigates approximations for computing MCR paths in the context of manipulation planning. The proposed framework searches for MCR paths up to a certain length of solution in terms of end-effector distance. This length can be defined as a multiple of the shortest path length in the space when movable objects are ignored. Given experimental evaluation on simulated manipulation planning challenges, the bounded-length approximation provides a desirable tradeoff between minimizing constraints, computational cost and path length.

Keywords: Shortest path; Constraint removal; Manipulation planning; Multiple objects; Rearrangement;

1. Introduction

The classical version of the path planning problem involves computing the shortest feasible path for a given start and goal. This is typically computed over an underlying state space, which is frequently abstracted as a graph. In many situations, however, there are constraints that cause no solution path to exist. In such scenarios, a complete path planner will keep searching for all possible alternative paths in the state space and, upon failure, it will exit reporting its inability to compute a solution. In many application domains, however, it may be possible to apply changes in the underlying state space so that a solution will become feasible.

In particular, consider robot manipulation and rearrangement challenges [1–5], such as the one shown in Figure 1. In this setup, a robotic arm needs to transfer an object from the left side of the shelf to the right side. This problem turns out to have no collision-free path, as the goal is not reachable given the placement of obstacles. A desirable behavior is for the manipulator to detect the lack of a solution and identify the minimum changes that it has to apply in the environment in order for the problem to become solvable. These changes can correspond to the transferring obstacles that can be removed from the scene.

Thus, a new problem can be formulated as a possible way to address such situations: what is the minimum number of movable obstacles that can be removed to make the problem feasible? This

[*]Kostas E. Bekris. Email: kostas.bekris@cs.rutgers.edu

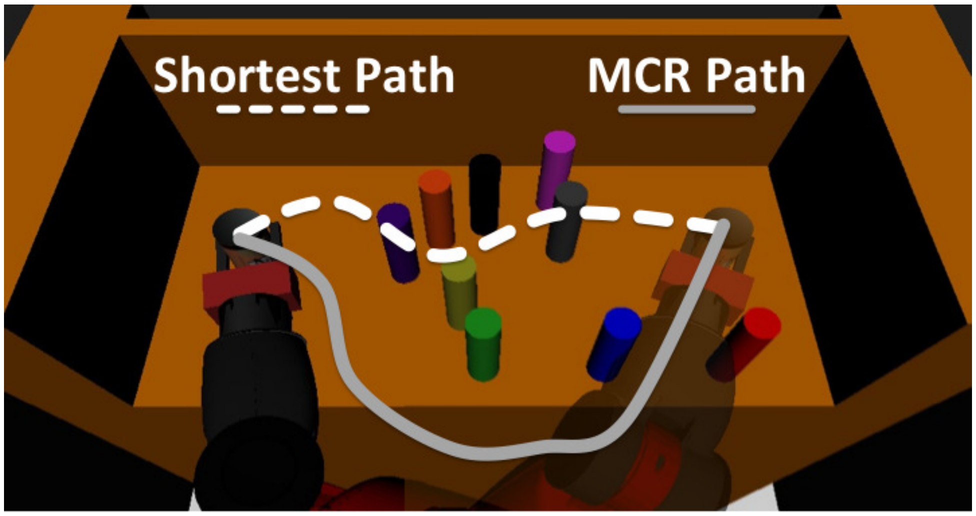

Figure 1. A minimum constraint removal problem in manipulation planning given a Baxter robot. Dashed line: The shortest path that ignores movable obstacles can be computed quickly but go over many constraints/movable obstacles. Solid line: The minimum constraint path will return the path with the minimum number of constraints but it is more expensive to compute.

problem has received attention recently and is referred to as the *minimum constraint removal* (MCR) problem [6, 7].

Beyond manipulation, there are many other application domains that can benefit from solutions to the MCR problem. In particular, it is helpful so as to provide minimal and humanly understandable explanations regarding the infeasibility of a challenge [8]. In multi-agent path finding[9], the MCR path of a moving agent can provide the minimum set of other agents that need to evacuate its path. Some of these algorithms do search in a way that reasons about constraints along solution paths [10]. In planning under uncertainty, paths that minimize collision probability could correspond to those that minimize the number of colliding volumes given a particle representation for detected objects in the world. Block sliding puzzles [11], such as the "Move it!" puzzle, may also involve MCR subproblems.

While important in many applications, the MCR challenge is harder than searching for the shortest path. The issue arises because MCR paths do not satisfy dynamic programming properties. In particular, a subset of an optimal solution is not necessarily an optimal solution itself. Examples of this complication are provided in this paper. Furthermore, MCR can be seen as a generalization of the task of determining the non-existence of a path between two points, which is known to be a hard challenge [12, 13].

This paper reviews possible solutions for the MCR problem: (i) a naive approach, that exhaustively searches all paths and is computationally infeasible, (ii) a greedy strategy, which is not guaranteed to return the correct solution but brings the promise of computational efficiency, (iii) an exact approach, which takes advantage of the problem structure to prune paths that cannot be solutions. Similar to existing work on the subject [6, 14], the experimental results show that while incomplete in the general case, the greedy strategy frequently computes solutions with the same number of constraints to be minimized as the exact approach. But the improvement in computational performance is shown here to be small in a robot manipulation setup.

Given the above techniques, this work proposes bounded path length approaches as an alternative for MCR[15]. The proposed methods bound the length of MCR paths they search for, given a multiple of the shortest path length in a constraint-free version of the space. The evaluation shows that the computational improvement is more significant relevant to the greedy approach, while the number of constraints found is close to what is found by the exact approach. Furthermore, the bounded-length approximation can also provide an anytime solution, where the bound on path length can incrementally increase, resulting in solutions of improving quality given the availability of additional computational resources. Relative to the earlier version of this work [15], the current paper provides additional evaluation, more extensive coverage of the related literature and more detailed description of the algorithms.

2. Background

Related challenges to the problem considered in this work are disconnection proving, excuse-making and minimum constraint removal problems. The applications of disconnection proving correspond primarily to feasibility algorithms that try to detect if a solution exists given certain constraints in the environment [12, 13, 16, 17]. Nevertheless, this is a computationally expensive operation, where practical solutions are limited to low-dimensional or geometrically simple configuration spaces. A related challenge is to consider excuse-making in symbolic planning problems. In these approaches the "excuse" is used to change the initial state to a state that will yield a feasible solution [8].

MCR formulations can be useful in the context of navigation among movable obstacles (NAMO), [18–20], as well as manipulation in cluttered environments [21], where it is necessary to evacuate a set of obstacles for an agent to reach its target. Such challenges were shown to be hard if the final locations of the obstacles are unspecified and PSPACE-hard when specified [20]. It is even NP-hard for simple instances with unit square obstacles [22]. Thus, most efforts have dealt with efficiency [23, 24] and provide completeness results only for problem subclasses [18, 19]. NAMO challenges relate to the Sokoban puzzle, for which search methods and proper abstractions have been developed [25, 26].

Similar work to the computation of MCR paths has addressed violating low priority tasks for multi-objective tasks specified in terms of `LTL` formulas [27]. An iteratively deepening task and motion planning method uses similar ideas in order to add and remove constraints on motion feasibility at the task level [28].

In the minimum constraint removal problem, the goal is to minimize the amount of constraints that have to be displaced in order to yield a feasible path [6, 14, 15]. Different approaches have been proposed in the related literature. A computationally infeasible approach that searches all possible paths, as well as a faster greedy, but incomplete strategy were presented together with the formulation of the problem in the context of robotics challenges [6, 14]. The minimum constraint removal problem is proven to be NP-hard, even when the obstacles are restricted to being convex polygons [7].

The focus of the authors' work has been on balancing computational efficiency with the capability of returning paths with a small number of constraints by bounding the length of the possible solutions relative to the shortest length path that ignores removable constraints [15]. Finding the Minimum Constraint Removal path (MCR) can be useful in many applications, such as rearranging objects using a manipulator [2, 29]. The idea is to identify the minimum set of obstacles to be cleared from the workspace to manipulate a target object.

3. Problem Setup and Notation

Frequently, motion planning challenges do not have a solution, given the presence of constraints in the environment. In scenarios where the path is blocked by movable objects, the minimum constraint removal problem asks for the minimum set of constraints, which if removed from the scene, they provide a feasible solution.

3.1 *Abstract Problem*

Consider a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ that represents the connectivity of a state space. Each node corresponds to a state of the manipulator, while an edge expresses a local trajectory between two states. The weight of an edge is set equal to the distance between the two nodes defining the edge in the manipulator's state space. Moreover, it is possible to define for each edge $e \in \mathcal{E}$ a set of constraints c_e. The objective is to compute a path on $\mathcal{G}$ that minimizes the number of constraints that the path traverses. Figure 2 describes a relevant setup.

The constraints along a path $\pi(v,u) = \{e_1,\ldots,e_n\}$ will be denoted as $\mathrm{c}(\pi(v,u))$ and correspond to the union of the constraints along the edges of the path: $\mathrm{c}(\pi) = \cup_i c_{e_i}$.

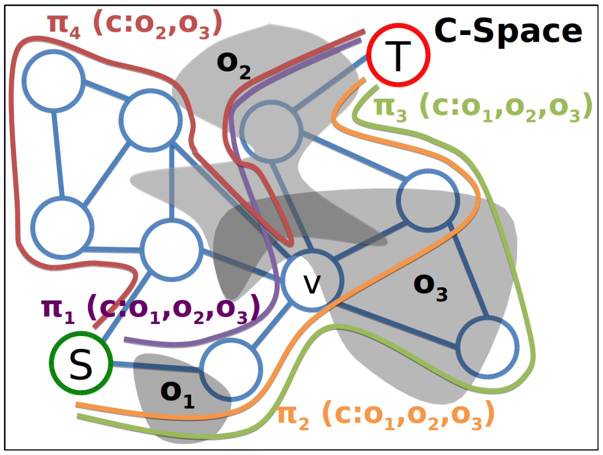

Figure 2. An example of a graph embedded in a space with constraints. The dark areas correspond to regions with constraints, which if removed they allow for a feasible path along the graph.

Minimum Constraint Removal Path: Consider a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, where for each edge $e \in \mathcal{E}$ a set of constraints c_e is defined. Then, given a start $s \in \mathcal{V}$ and a target node $t \in \mathcal{V}$, compute the minimum constraint removal path $\pi = \{e_1,\ldots,e_n\}$ on $\mathcal{G}$ that connects s and t, so that the number of unique constraints on the path $\mathrm{c}(\pi) = \cup_i c_{e_i}$ are minimized over all paths between s and t.

3.2 *The case of manipulation*

Consider the following setup in a 3D workspace:

- A robotic manipulator, that is able to acquire configurations $q \in \mathcal{Q}$, where $\mathcal{Q}$ is the manipulator's configuration space.
- A set of static obstacles $\mathcal{S}$. Given the presence of $\mathcal{S}$ it is possible to define the subset of $\mathcal{Q}$ that does not result in collisions with the static obstacles: $\mathcal{Q}_{free}$.
- A set of movable rigid-body objects $\mathcal{O}$, where each object $o_i \in \mathcal{O}$ can acquire a pose $p_i \in SE(3)$.
- A target object o, which is located in a starting pose $p^s \in SE(3)$ and needs to be transferred to a target pose $p^t \in SE(3)$. The target object o does not belong in the set $\mathcal{O}$.

Given the pose p of object o, it is possible to define a grasping configuration $q(p) \in \mathcal{Q}$ for the manipulator. For instance, this can be achieved through the use of inverse kinematics. The underlying manipulation challenge considered here is to find a path for the robot manipulator, which starts from a given grasping configuration $q(p^s) \in \mathcal{Q}_{free}$ for the start pose p^s of object o and transfers the object to a target pose p^t given a grasping configuration $q(p^t) \in \mathcal{Q}_{free}$. The relative pose between the robot's end-effector and the object - i.e., the grasp - is the same given the arm configuration/object pose pairs $(q(p^s), p^s)$ and $(q(p^t), p^t)$, so no in-hand manipulation is necessary to transfer the object.

Beyond the static geometry, the problem is further complicated by the presence of the movable objects $\mathcal{O}$. Each object $o_i \in \mathcal{O}$ at pose p_i defines a subset of manipulator configurations $\mathcal{Q}^{o_i} \subset \mathcal{Q}_{free}$ that result in a collision between object o_i and the manipulator or the transferred object o given the grasp. The focus is on situations where there is no solution path that takes the manipulator carrying the object o from $q(p^s)$ to $q(p^t)$ without intersecting any of the sets $\mathcal{Q}^{o_i}$, where $o_i \in \mathcal{O}$. In this context, the objective is then adapted so as to identify the minimum set of movable objects o_i that need to be removed from the scene so as to be able to solve the original manipulation objective.

To deal with this objective and along the lines of the abstract MCR problem definition, this work assumes that a graph structure in the collision-free configuration space $\mathcal{Q}_{free}$ of the manipulator, i.e., a roadmap, is computed first in order to compute paths for the manipulator. This can be done either by using sampling-based planners that sample nodes and edges of a roadmap in the configuration space, such as with a PRM approach or other sampling-based planners [30–33], or search-based methods that implicitly consider a discretization of the configuration space and directly search over it [1].

Such a roadmap for the manipulator can be seen as the equivalent to the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ considered in the abstract MCR definition. The nodes $\mathcal{V}$ of the roadmap correspond to configurations of the manipulator in $\mathcal{Q}_{free}$ and the edges $\mathcal{E}$ to straight-line paths in $\mathcal{Q}_{free}$ that connect pairs of nodes. The roadmap can be used to search for transfer paths for the object o by placing the object at the end-effector given the grasp defined according to $q(p^s)$ to $q(p^t)$ during query resolution. Then, traversing an edge e of the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, corresponds to a sequence of motions for the manipulator carrying the object o, which are collision-free with the static geometry $\mathcal{S}$ but may intersect a subset of the $\mathcal{Q}^{o_i}$ configuration sets.

Intersections with one of the $\mathcal{Q}^{o_i}$ along an edge $e \in \mathcal{E}$ are equivalent to the edge e having a constraint that needs to be avoided in the context of MCR. The set of constraints of an edge e of the roadmap will be denoted as c_e, and correspond to the set of objects in $\mathcal{O}$ that cause a collision with the manipulator or the carried object o along the edge e. An example is shown in Figure 2, where a graph is embedded in the configuration space $\mathcal{Q}_{free}$ and the gray regions correspond to constraints arising from different movable obstacles.

Then, similar to the abstract problem, the objective is to find the path $\pi = \{e_1, \ldots, e_n\}$ along the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ storing manipulator configurations, which has the minimum number of constraints. This path will correspond to the minimum number of objects $\mathcal{O}$ that need to be removed in order for the manipulator to be able to move the object o_i from pose p^s to pose p_i^t.

Note that a solution to this MCR problem can be easily applied both to the computation of transit and transfer paths. In transit problems the manipulator is not holding an object and needs to move between two configurations in $\mathcal{Q}_{free}$. The experimental evaluation of this work includes both transit and transfer challenges.

4. Search for Minimum Constraint Removal Paths

The basic framework for finding MCR paths corresponds to a best-first search methodology over the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, which makes use of a priority queue Q. The priority queue holds search elements $u = \{\mathrm{v}, \pi, c, f\}$, which correspond to the following information:

- $u.\mathrm{v}$: The corresponding graph node $u.\mathrm{v} \in \mathcal{V}$ for this search element.
- $u.\pi$: The path from the start graph node s to graph node $u.\mathrm{v}$ corresponding to this search element. The length of the path is denoted as $|u.\pi|$.
- $u.c$: The set of constraints along the path $u.\pi$, which corresponds to a sequence of edges $\{e_1, \ldots, e_n\}$ from the set $\mathcal{E}$. The set of constraints for π is the union of individual constraints along the edges of the path $u.\pi$, i.e., $c_\pi = \bigcup_{\forall e_i \in \pi} c_{e_i}$.
- $u.f$: An evaluation function for the search element, which typically depends on the length of the path $|u.\pi|$ and potentially a heuristic estimate $h(u.\mathrm{v}, t)$ of the length of the shortest path from $u.\mathrm{v}$ to the target graph node t. For instance, for uniform-cost search, the evaluation function will be $u.f = |u.\pi|$ but for A^* it will be $u.f = |u.\pi| + h(u.\mathrm{v}, t)$.

4.1 *Best-First Search for Finding the Shortest Path*

Consider first the traditional objective of computing the shortest path on graph $\mathcal{G}$ and ignoring the constraints $u.c$ of the search elements. This base case will be used to emphasize what is different when the constraints are taken into account and make clear the experimental results, which include an implementation of a shortest-length path algorithm as a comparison point.

In this setup, the priority queue is using only the evaluation function $u.f$ to order search elements. During each iteration of the algorithm, the top search element u_{top} is removed from the priority queue Q and expanded. The expansion step considers the neighbors v_{neigh} of node $u_{top}.\mathrm{v}$ on the graph, i.e., $\mathrm{v}_{neigh} \in Adj(\mathcal{G}, u_{top}.\mathrm{v}_{neigh})$. For the neighbors v_{neigh}, there is an edge $e(u_{top}.\mathrm{v}, \mathrm{v}_{neigh})$ on the set of edges $\mathcal{E}$. Denote as c_e the constraints of this edge. Every time that a neighbor v_{neigh} is evaluated, one of following happens:

- The graph node v_{neigh} has not been previously encountered, in which case a new search element u_{neigh} is added to the queue, which corresponds to the following tuple:
  - the graph node v_{neigh};
  - the path $u_{neigh}.\pi = u_{top}.\pi | e(u_{top}.\mathrm{v}, \mathrm{v}_{neigh})$, where the operand $|$ denotes concatenation;
  - the set of constraints $u_{top}.c \cup c_e$;
  - and the evaluation function $u.f = |u_{neigh}.\pi| + h(\mathrm{v}_{neigh}, t)$.
- There is already a search element u_{neigh} in the queue that stores graph node v_{neigh}, in which case:
  - if the path $u_{neigh}.\pi$ on the search element is longer than the path to v_{neigh} via $u_{top}.\mathrm{v}$, then the search element is updated to store the shorter alternative path; similarly, the constraints and evaluation function get updated and the queue is resorted;
  - otherwise, the queue is not affected.

The above process results in having an efficient algorithm and relates to the dynamic programming structure of shortest length paths. In other words, every time a new path to an already visited node v_{neigh} is discovered that is longer than an existing path, it is known that this cannot be part of the optimal path to the target t.

4.2 *Exhaustive and Greedy Search for* MCR

At first glance, it may appear that the exact same process can be used to compute MCR paths. The MCR version of the algorithm would require alterations to the ordering of the priority queue and the criterion that determines when new search elements are added to the queue. Instead of the evaluation function $u_{top}.f$, it is possible to order elements based on the number of constraints $|u_{top}.c|$ and promote the selection of search elements that have a small number of constraints. Ties can be broken by making use of the evaluation function $u_{top}.f$. Furthermore, every time that a neighboring node v_{neigh} is reached, for which a search element u_{neigh} already exists in the queue, if the number of constraints in the existing search element $u_{neigh}.c$ is less than those of the newly discovered path via u_{top}, then the new path is discarded.

This paper, as previous work on the topic [6], refers to the above methodology as a "greedy" approach for computing MCR paths. The greedy algorithm will prune a newly discovered path π' to node v when $|\mathrm{c}(\pi'(s,v))| \geq |\mathrm{c}(\pi(s,v))|$, where π is an already discovered path to v.

Unfortunately, the dynamic programming property is not true if the objective is to compute the path with the minimum set of constraints from s to t. The issue is illustrated in Figure 3. Consider the two highlighted paths from s to node v. The first one π_1 goes through only one constraint o_1. The second one π_2, goes through two constraints o_2 and o_3. Evidently, the optimal MCR path up to node v among the two is π_1. This means, that if the above greedy best-first process is used, once π_2 is considered at node v it will be discarded.

Notice, however, that all paths from node v to node t go through the constraints o_2 and o_3. This means that if one considers extending path π_2 to node t, the final number of constraints will be 2, corresponding to the constraints o_2 and o_3. If one extended path π_1 to node t, the final number of constraints will be 3, corresponding to all three constraints in this space. Thus, the optimal solution to node t is the one that extends path π_2. But the greedy search procedure prunes path π_2 at node v because locally it is suboptimal. In this way, it is not able to discover

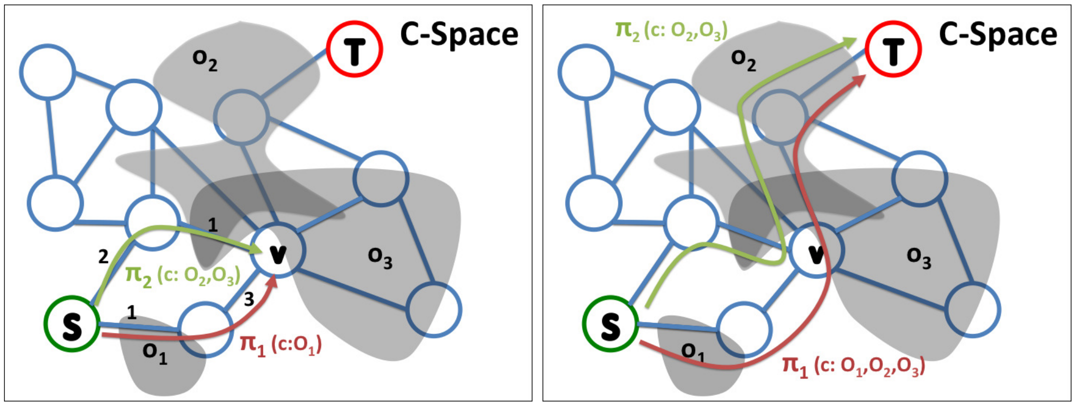

Figure 3. *Left:* Two paths, π_1 and π_2, from s to node v. π_2 has two constraints: o_2, o_3, while π_1 has only one: o_1. At node v, the optimum path is π_1. *Right:* Two paths to the target node, which are extensions of π_1 and π_2. The optimum MCR solution is the extension of path π_2, with only two constraints versus three for the alternative.

the globally optimal path to the target node t.

One naïve complete solution that addresses the above issue is to exhaustively consider all simple paths in the space, i.e., those without loops, as shown in Figure 2. This means that every time a neighbor v_{neigh} is discovered by the best-first procedure, a corresponding search element is always added to the queue Q, unless v_{neigh} is already included in the path to u_{top}.v. Obviously, this is a very expensive process that does not scale well, since there is an exponential number of paths to the target node. Nodes of the graph will have to be expanded multiple times in order to compute all the possible paths to each node.

There is no experimental evaluation of the "naïve" solution accompanying this paper as the running time is orders of magnitude slower than alternatives. The greedy approach is evaluated, however, and it is shown that it can frequently find satisfactory solutions.

4.3 *More Efficient Exact Search*

There is an alternative to the naïve complete solution, which performs some pruning of paths and is able to find the exact number of minimum constraints.

As it has been argued above, a suboptimal path to a node v may be a subset to an optimal path at the target node t (Fig.3). Not all suboptimal paths, however, can be subsets of optimal solutions. In particular, if the set of constraints of a path is dominating the set of constraints of another path to the same node, then the first one cannot possibly be a subset to an optimal solution. This means that a new path π' approaching v will be pruned only if the set of the constraints of π' is super-set of the constraints of a path π already reaching node v, i.e., if $\mathrm{c}(\pi(s,\mathrm{v})) \subset \mathrm{c}(\pi'(s,\mathrm{v}))$. If the set of the constraints is exactly the same, the new path approaching v will be pruned only if the length of the path will be longer than previously discovered.

This algorithm will also result in expanding multiple times the same node of the graph. This is because paths with different combination of constraints can reach each node v. Although, it is slower than the greedy algorithm, it is guaranteed to return the MCR solution and is faster than the exhaustive search because it prunes dominated paths. For example, in Figure 3 the algorithm will not prune away any of the π_1 and π_2 paths, since none dominates the other. As a result, node v will be expanded twice. In this way, the algorithm will detect that the suboptimal path π_2 will eventually yield the true optimal MCR solution path.

Algorithm 1 describes the EXACT_MCR approach and the overall best-first framework. The algorithm uses a priority queue, which prioritizes search elements that have a low number of constraints $|c|$. Ties are broken in favor of elements with a small evaluation function f. The queue is initialized with a search element corresponding to the start node (line 1).

While there are nodes in the queue (line 2), the algorithm will pop the highest priority search

Algorithm 1: EXACT_MCR($\mathcal{G}(\mathcal{V},\mathcal{E}), s, t$)

```
1  $Q \leftarrow$ NEW_ELEMENT$(s, \emptyset, \emptyset, 0)$;
2  while $Q$ not empty do
3      $u_{top} \leftarrow Q.pop()$;
4      if $u_{top}.\mathrm{v} == t$ then
5          return $u_{top}.\pi$;
6      for each $\mathrm{v}_{neigh} \in Adj(\mathcal{G}, u_{top}.\mathrm{v})$ do
7          $\mathrm{c} \leftarrow u_{top}.\mathrm{c} \cup e(u_{top}.\mathrm{v}, \mathrm{v}_{neigh}).\mathrm{c}$;
8          if IS_NEW_SET$(\mathrm{v}_{neigh}.\mathbf{C}, \mathrm{c})$ then
9              $\pi \leftarrow u_{top}.\pi \mid e(u_{top}.\mathrm{v}, \mathrm{v}_{neigh})$;
10             $f \leftarrow |\pi| + h(\mathrm{v}_{neigh}, t)$;
11             $Q \leftarrow$ NEW_ELEMENT$(\mathrm{v}_{neigh}, \pi, \mathrm{c}, f)$;
12     return $\emptyset$;
```

element u_{top} from Q (line 3) and it will check if the corresponding graph node is equal to the target node t (line 4). If the nodes are equal then the algorithm will return the path that is stored in the search element (line 5). Otherwise, all the adjacent graph nodes v_{neigh} of $u_{top}.\mathrm{v}$ will be considered (line 6).

The algorithm will first generate the new set of constraints corresponding to the path to v_{neigh} via $u_{top}.\mathrm{v}$ (line 7). Then, the function IS_NEW_SET will check if the new set of constraints c is a subset of any of the constraints of paths that have already been generated for node v_{neigh}. In order to speed up implementation, each graph node can keep track of the different sets of constraints, $\mathbf{C}$, for each path that has reached the node, and the corresponding evaluation functions f. The operation of the approach IS_NEW_SET is the following:

- If there is a constraint set $\mathrm{c}' \in \mathbf{C}$, where $\mathrm{c}' \subset \mathrm{c}$, then the algorithm will return false and the node will not be added in the queue Q.
- If there is a constraint set $\mathrm{c}' \in \mathbf{C}$, where $\mathrm{c} \equiv \mathrm{c}'$, then the algorithm will compare the f values. If the new f' value is smaller than the existing one, IS_NEW_SET will return true.
- If there is a constraint set $\mathrm{c}' \in \mathbf{C}$, where $\mathrm{c} \subset \mathrm{c}'$, then the set c' will be replaced by c and the algorithm will return true.
- If none of the above is true, then the algorithm will return true and c will be added in the $\mathbf{C}$ list of the graph node.

If IS_NEW_SET decides that the constraints set is a new set of constraints, then the algorithm will create a new search element and will update the corresponding graph node with the new information, using the NEW_ELEMENT function (lines 9-11).

4.4 *Bounded-Length Search*

In most applications, and specifically in the manipulation setup considered in this paper, the search space is significantly larger than the space that the approach needs to search in order to find a reasonable solution to the problem. The previously described algorithms can waste a lot of time searching away from the solution before searching along the target. Figure 4 depicts a scenario where both the greedy algorithm and the exact algorithm will waste a lot of time searching over the collision-free edges before moving through the objects towards the target. Note that this situation arises frequently in manipulation, where a target object may be located in a shelf or a cabinet.

This paper proposes an approximate method that is faster than both the exact and greedy algorithms, while balancing a trade-off in terms of the number of constraints identified. In par-

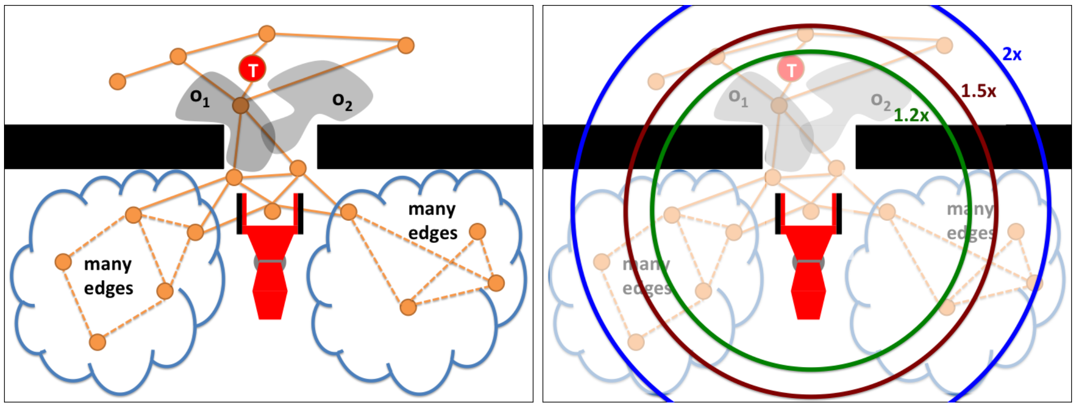

Figure 4. *Left:* A case where the Bounded-Length algorithm will not waste a lot of time to check the collision free areas, before moving towards the target.*Right:* If there is more time available for searching a path with less constraints could be detected by increasing the searching area.

ticular, the Bounded-Length version of the EXACT_MCR approach is shown in Algorithm 2. This version allows to incrementally increase a threshold on the length of the path discovered by the search process until there is convergence to the true optimal MCR. The threshold is computed based on the shortest path ignoring any constraints and a multiplying factor. For instance, if the length of the shortest path, which ignores the effects of constraints (i.e., the movable obstacles) is X, and the multiplying factor is F, then the algorithm will search only paths of length up to $(F \times X)$. It is straightforward to consider an incremental search approach where the multiplying factor F is incrementally increased resulting in an anytime solution for the computation of minimum constraint removal paths.

Algorithm 2: BL_MCR$(\mathcal{G}(\mathcal{V}, \mathcal{E}), s, t, threshold)$

```
1  $Q \leftarrow$ NEW_ELEMENT$(s, \emptyset, \emptyset, 0)$;
2  while $Q$ not empty do
3      $u_{top} \leftarrow Q.pop()$;
4      if $u_{top}.\mathrm{v} == t$ then
5          return $u_{top}.\pi$;
6      for each $\mathrm{v}_{neigh} \in Adj(\mathcal{G}, u_{top}.\mathrm{v})$ do
7          $\pi \leftarrow u_{top}.\pi \mid e(u_{top}.\mathrm{v}, \mathrm{v}_{neigh})$;
8          if $|\pi| < threshold$ then
9              $\mathrm{c} \leftarrow u_{top}.\mathrm{c} \cup e(u_{top}.\mathrm{v}, \mathrm{v}_{neigh}).\mathrm{c}$;
10             if IS_NEW_SET$(\mathrm{v}_{neigh}.\mathbf{C}, \mathrm{c})$ then
11                 $f \leftarrow |\pi| + h(\mathrm{v}_{neigh}, t)$;
12                 $Q \leftarrow$ NEW_ELEMENT$(\mathrm{v}_{neigh}, \pi, \mathrm{c}, f)$;
13     return $\emptyset$;
```

Algorithm 2 works similar to Algorithm 1 with the difference that the new algorithm first checks in lines 7-8 if the path has length greater than the threshold. If the path is within the threshold, the BL_MCR approach will execute the same steps as in EXACT_MCR. The proposed algorithm with a small threshold will be able to search through short length paths that violate constraints fast without concentrating time searching through constraint-free nodes far from the shortest length paths. The algorithm cannot guarantee to return the optimum MCR path for a fixed length search. In practice, however, it returns good solutions, close to those of the exact approach for MCR and with paths that are of known length relative to the shortest.

In the implementation accompanying this paper, the threshold is provided as input to the algorithm. The corresponding path quality and achieved constraints are reported. As indicated in the paper, it is straightforward to implement an incremental search approach where the multiplying factor is incrementally increased (Figure 4(right)) resulting in an anytime solution for the computation of minimum constraint removal paths. Similarly, it is also possible to execute the algorithm by specifying as input only the available computation time. In this case, the algorithm can maintain all the nodes in the queue and expand only those nodes that are within a certain threshold. If the queue does not have more nodes to expand below the threshold and if there is more time available, then the threshold is increased and the search continues with the algorithm popping from the queue additional nodes on the frontier of the search space. Experiments show that thresholds in the order of 1.5 times the length of the shortest path return a path close to optimal minimum constraint path, but two orders of magnitude faster than the exact algorithm.

5. Evaluation

The methods have been tested in the setup of Figure 1, as well as in two different scenarios in a cluttered shelf with limited maneuverability as in Figures 5 and 6. A model of a Baxter arm is used for testing. For the initial benchmark environment, the Baxter arm has to move a target object from the left side to the right side, while 9 objects are blocking the straight transfer path for the arm. For the shelf, 10 to 20 cylinders are placed randomly. In the first variation of the shelf challenge, the arm will start outside the shelf (Fig.5 a,b) and try to detect the minimum number of cans that it has to move in order to transit and grasp a beer can at the back of the shelf.

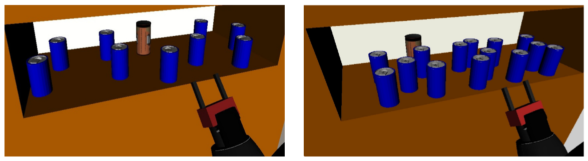
(a) Approach: 10 objects (b) Approach: 20 objects

Figure 5. A setup for the "transit" scenario inside a shelving unit. The problem is to compute the minimum constraint removal path that allows the robotic arm to move from outside the shelf until a grasping configuration for the beer. (a) 10 objects or (b) 20 objects are present in the shelf and are blocking the manipulator's path.

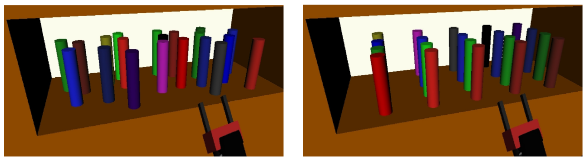
(a) Transfer: Initial state (b) Transfer: Final state

Figure 6. A setup for the "transfer" scenario inside a shelf, where (a) provides the initial arrangement of the objects and (b) provides the final one. For this benchmark, the algorithm is called as many times as the number of objects in the scene (in the figure, the case of 20 object is displayed). For each object, the objective is to compute the minimum constraint removal path from a grasping configuration of the arm at the object's initial pose to a grasping configuration of the arm at the object's final pose. All other objects are assumed to be at their initial location during the computation of the path.

For the second scenario the objective is to find the MCR paths for the objects to be placed on a grid (Fig.6 a,b). The Baxter arm starts from a state where it grasps one of the objects in its initial grasping configuration $q(p^s)$. The algorithm will be called to compute a path for the arm to place the object in its target pose at configuration $q(p^t)$. Note that in this scenario there is a high probability for the arm to be in collision at the initial state.

Four methods are tested: (a) shortest length path, (b) the greedy algorithm, (c) the exact search approach and (d) the bounded-length version of the exact approach with different values for the threshold. The numbers $1.3, 1.5, 2$ after the name of the bounded-length algorithm correspond to the different multiplying factors for computing the threshold. As indicated above, the threshold is a multiple of the length of the shortest path.

Taking advantage of object symmetry in the corresponding experimental setup, five parallel grasps were randomly sampled for each cylindrical object. The grasps are such so that the midpoint between the two fingers of the parallel gripper is placed at the center of the object. Furthermore, the orientation of the gripper is always perpendicular to the axis of the cylindrical object. This leaves one degree of freedom undefined, which is the angle of the gripper about

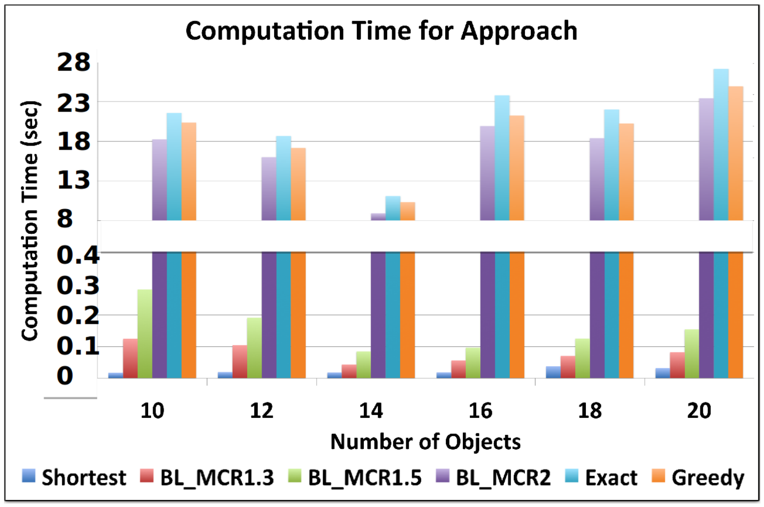

Figure 7. Computation time for the transit scenario in the shelf environment for all the algorithms.

the object's axis. If the angle 0 corresponds to the approach of the gripper from the side of the object facing outside the shelf, then this value is sampled from the range of $[-1, 1]$ radii. 50 experiments were performed for each combination of method and environment. The shortest length path algorithm ignores the movable objects. It computes the shortest length path and the number of constraints are reported for comparison.

Table 1. Benchmark

Variables	*Time*	*Constraints*
SHORTEST_PATH	0.093	7
BL_MCR1.3	0.195	7
BL_MCR1.5	0.201	5
BL_MCR2	0.454	5
BL_MCR3	0.814	3
EXACT_MCR	1.86	3
GREEDY_MCR	1.552	3

Average computation time and the number of constraints for all the algorithms in the initial benchmark environment of Figure 1.

A transfer and a transit roadmap have been precomputed for this challenge and they contain on each edge the set of objects poses that lead to collisions. This allowed to speed up the execution of multiple experiments so that each path is computed in sub-second time. If this preprocessing is not available, then the computation time for each algorithm is approximately two orders of magnitude larger, since collision checking needs to be performed online. But this change affects all algorithms uniformly. Furthermore, a manipulation algorithm for rearranging all the objects from their initial (Fig.6a) to their target (Fig.6b) poses, needs to make thousands of MCR calls. This means that small changes in the running time of MCR approach can have a significant impact on the cost of an object rearrangement solution.

Table 1 shows the results of a single run for the benchmark environment (Fig.1). In this example, the manipulator has to find a path to grasp the object, transfer the object to its target pose and finally return to its initial state. For this task, if the manipulator is using the shortest path while ignoring constraints, then it has to go through seven movable objects in the environment.

When the proposed bounded length algorithm is used with a path length multiplier of 1.3, then the returned path still includes seven constraints as in the case of the shortest path. By increasing the multiplier to 1.5 or 2 the algorithm is able to find a better path with fewer constraints. It can do so faster than both the greedy and the exact algorithms, which are able

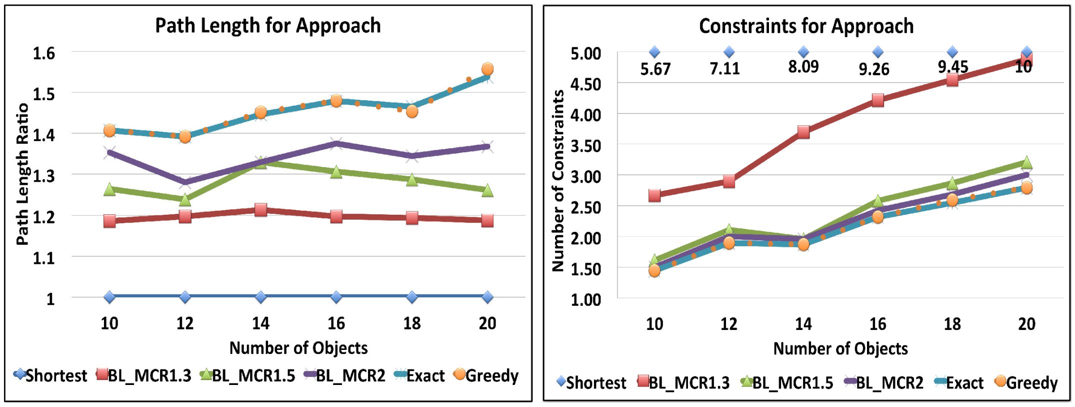

Figure 8. *Left:* Path length ratio relative to the shortest length path discovered for all the algorithms for the transit scenario in the shelf environment. *Right:* The average number of constraints for each algorithm in the transit scenario in the shelf.

to detect a solution with only three constraints. The exact method has to search more before it finds a solution relative to the greedy, which manages to expand fewer nodes than the exact method. The greedy solution, however, is slower than the proposed bounded length one. If the threshold for the bounded-length variant is increased even further, e.g., for a multiplier of 3, then the same number of constraints as the exact method will be discovered in 0.814 seconds, which is still twice as fast relative to the exact and greedy algorithms. Overall, the trade-off provided by the proposed solution clearly arises. As the threshold for the bounded-length version becomes smaller, the algorithm becomes faster in finding a solution but returns additional constraints.

The first scenario in the shelf environment examines the case where a set of movable objects blocks reaching a target object. The task for the first test is to find a path with the minimum number of constraints in order to move the manipulator from its initial state to grasp the brown object. For this test a probabilistic roadmap is built using uniform sampling. This typically results in a graph that has more nodes outside the shelf and fewer inside the shelf, given the narrow environment inside the shelf. As explained before, the exact and greedy algorithms will have to search the collision free graph before start considering paths with constraints. The computation time results of Figure 7 show again the computational advantages of the bounded-length solutions. The shortest length path does not check for collisions with the movable objects and returns solutions faster than alternatives. For all the other algorithms, the computation time increases. The exact method takes the longest time. Although the greedy algorithm expands fewer nodes, the computation time is almost the same. Nevertheless, the bounded-length variants result in faster computation of the solution. The difference in time between the algorithms relates to the issue described in Figure 4. The greedy and the exact algorithm will waste a lot of time searching over the collision-free edges before moving through the objects, a situation that arises often in manipulation. The proposed method will avoid expanding nodes that are far away from the target. In summary, it appears that a threshold of 1.5 results in solutions with a similar number of constraints as the exact algorithm but considerably faster.

In the context of the same challenge, Figure 8(right) shows that the shortest length path returns a significant number of constraints, although it is the shortest in length according to Figure 8(left). The remaining algorithms return fewer constraints and close to those of the exact solution, with the exception of BL_MCR using the smallest threshold. As the threshold is relaxed, the number of constraints approaches that of the exact. The exact and the greedy methods return the longest paths with the fewer constraints. The difference between the exact and the greedy is more visible when the manipulator is colliding with the same obstacle in different parts of a path. Given that the objects in this example are relatively thin, the arm collides with each object on average once, as a result these two algorithms do not have a big difference. Figure 8(left) depicts the ratio of the path lengths with respect to the shortest path. As the threshold for the bounded length version decreases, the path returned is closer to the shortest path, but

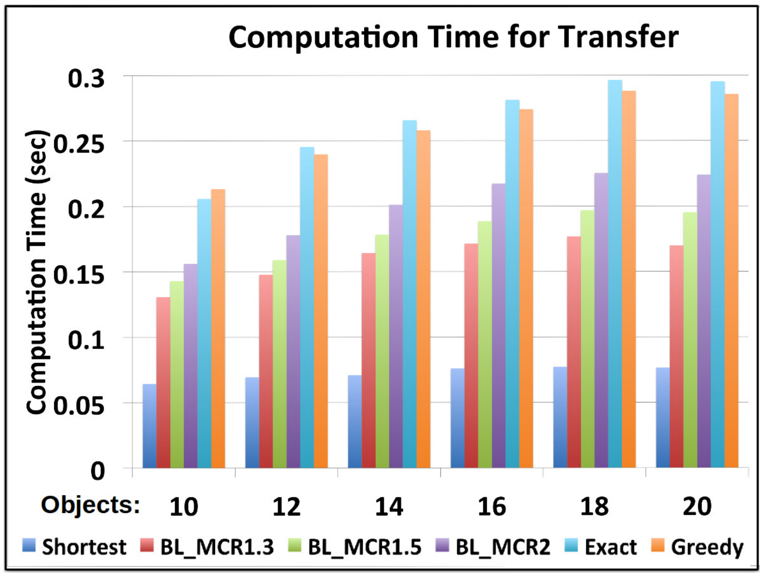

Figure 9. Computation time for the transfer scenario in the shelf environment for all the algorithms. The number of objects corresponds to how many movable objects are on the shelf.

generates more constraints.

The task for the second test is for the manipulator to move an object that it is already grasped from an initial pose to a target pose in a way that minimizes constraints, i.e., minimizes collisions with other objects in the environment. For the second test, a biased sampling approach is used for the construction of the probabilistic roadmap so as to increase the density of nodes inside the shelf. This helps solution times for the various methods relative to the first test case. The computation time for the second test in the shelf is provided in Figure 9. Similar results with the previous scenarios are achieved: the shortest length path returns solutions faster than the alternatives, the greedy and the exact are the slowest, while the BL_MCR achieves better computation time. In this example most of the graph is within the boundaries of the shelf in order to be able to connect initial and target poses for the objects inside the shelf. There is only a small part of the graph outside the shelf. Although the exact and the greedy algorithms do not have to search outside of the shelf given this setup, they are still the slowest algorithms to come up with a solution.

The exact and the greedy methods are not only taking more time to find a solution, they also return the longest paths. Figure 10 *Left:* depicts the ratio of the paths with respect to the shortest path. As the threshold for the bounded length version decreases, the path returned is closer to the shortest path. This results in more constraints for the returned path as Figure 10 *Right:* shows, but significantly lower than the number of constraints on the shortest path. The shortest length path involves a lot of constraints. Again, as before, as the solutions approach the shortest length path, they result in an increased number of constraints.

Combining the above results for the shelf, it appears that a threshold of 2 tends to return solutions with a similar number of constraints as the exact algorithm in this setup but faster. In particular, the 2 times bounded-length version returned solutions faster across all experiments for this benchmark with better path quality and a similar number of constraints to the exact and the greedy solutions.

6. Discussion

This work studies the computation of "minimum constraint removal" (MCR) paths, which can impact robot manipulation planning challenges. This is a computationally hard problem in the general case as such paths do not exhibit dynamic programming properties. Various algorithmic

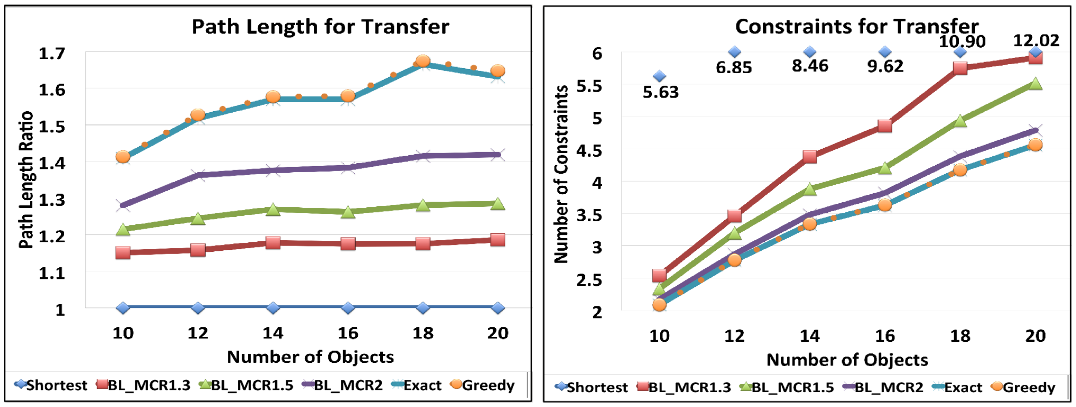

Figure 10. *Left:* Path length ratio relative to the shortest length path for all the algorithms in the transfer scenario. *Right:* The average number of constraints for each algorithm in the transfer scenario.

alternatives are described in this paper, varying from approximate solutions to exact algorithms. Approximate solutions that bound the path length of the considered path seem to provide a desirable trade-off in terms of returning solutions with a low number of constraints, relatively short path lengths and low computation time.

The description in this paper did not discuss what are the grasps and manipulation paths that can be used in order to remove the movable objects. Some of them may not be directly removable and this may result in an iterative computation of MCR paths. Nevertheless, the proposed approximate methods for the computation of MCR paths have been used in the context of more general rearrangement task planning solutions [2], where they have been shown to be benefit the computation of scalable paths for rearranging many objects.

The considered solutions can also be applied in the context of pick-and-place paths for general objects, where the arm first identifies the grasp necessary to pick up an object and transfer it to a desired target pose. In this case, the additional complication that can arise is that there may be many valid grasps at p^s and p^t, which may result to a different number of objects that need to be removed so that the problem becomes solvable. Iterating over different grasps and then using the solutions for the MCR problem discussed here is one way to solve such challenges.

It is interesting to compare against a different version of the problem, which satisfies dynamic programming principles, such as treating the movable obstacles as soft constraints that only introduce an increased cost for the corresponding edges but do not invalidate them. In this version of the problem, the choice of the cost for the soft constraints is rather arbitrary and can result in a variety of solutions. Furthermore, one could consider the use of trajectory optimization methods in this context. Such solutions could be used to initialize the global search for the minimum constraint removal path.

Acknowledgements

The authors are with the Computer Science Dept. at Rutgers University, NJ, USA. Their work is supported by NSF awards IIS-1617744, IIS-1451737, CCF-1330789. Any opinions, findings and conclusions or recommendations expressed in this paper do not necessarily reflect the views of the sponsors.

References

[1] Cohen JB, Chitta S, Likhachev M. Single- and dual-arm motion planning with heuristic search. International Journal of Robotic Research. 2014;33(2):305–320.

[2] Krontiris A, Bekris KE. Dealing with difficult instances of object rearrangement. In: Robotics: Science and Systems (RSS). Rome, Italy. 2015 July.

[3] Krontiris A, Bekris KE. Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In: International Conference on Robotics and Automation (ICRA). Stockholm, Sweden. 2016 05/2016.

[4] Krontiris A, Shome R, Dobson A, Kimmel A, Bekris KE. Rearranging similar objects with a manipulator using pebble graphs. In: IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS). Madrid, Spain. 2014 11/2014.

[5] Shuai H, Stiffler N, Krontiris A, Bekris KE, Yu J. High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods. In: Robotics: Science and Systems (RSS). Cambridge, MA. 2017 07/2017.

[6] Hauser K. The minimum constraint removal problem with three robotics applications. The International Journal of Robotics Research. 2013;.

[7] Erickson LH, LaValle SM. A Simple, but NP-Hard Motion Planning Problem. In: AAAI Conference on Artificial Intelligence. 2013.

[8] Gobelbecker M, Keller T, Eyerich P, Brenner M, Nebel B. Coming up with good excuses: What to do when no plan can be found. In: International Conference on Automated Planning and Scheduling. 2010.

[9] Sharon G, Stern R, Felner A, Sturtevant N. The Conflict-based Search Algorithm for Multi-Agent Pathfinding. Artificial Intelligence Journal (AIJ). 2015;:40–66.

[10] Sharon G, Stern R, Felner A, Sturtevant N. Conflict-based Search for Optimal Multi-agent Path Finding. AAAI. 2012;.

[11] Hearn R, Demaine E. P-Space Completeness of Sliding-block Puzzles and other Problems through the Non-Deterministic Constraint Logic Model of Computation. Theoretical Computer Science. 2005; 343(1):72–96.

[12] Zhang L, Kim Y, Manocha D. A Simple Path Non-Existence Algorithm using C-Obstacle Query. In: Workshop on the Algorithmic Foundations of Robotics (WAFR). 2008.

[13] McCarthy Z, Bretl T, Hutchinson S. Proving Path Non-Existence using Sampling and Alpha Shapes. In: IEEE International Conference on Robotics and Automation (ICRA). 2012. p. 2563–2569.

[14] Hauser K. Minimum Constraint Displacement Motion Planning. In: RSS. 2013.

[15] Krontiris A, Bekris KE. Computational tradeoffs of search methods for minimum constraint removal paths. In: Symposium on Combinatorial Search (SoCS). Dead Sea, Israel. 2015 June.

[16] Basch J, Guibas LJ, Hsu D, Nguyen AT. Disconnection proofs for motion planning. In: IEEE International Conference on Robotics and Automation. Vol. 2. 2001. p. 1765–1772.

[17] Bretl T, Lall S, Latombe JC, Rock S. Multi-step Motion Planning for Free-Climbing Robots. In: WAFR. 2004.

[18] Stilman M, Kuffner J. Navigation among Movable Obstacles: Realtime Reasoning in Complex Environments. In: Humanoid Robotics. 2004. p. 322–341.

[19] Stilman M, Kuffner JJ. Planning Among Movable Obstacles with Artificial Constraints. In: WAFR. 2006.

[20] Wilfong G. Motion Planning in the Presence of Movable Obstacles. In: Annual Symp. of Computational Geometry. 1988. p. 279–288.

[21] Dogar M, Srinivasa S. A Push-Grasping Framework in Clutter. In: RSS. 2011.

[22] Demaine E, O'Rourke J, Demaine ML. Pushpush and push-1 are NP-hard in 2D. In: Canadian Conf. on Computational Geometry. 2000. p. 211–219.

[23] Chen PC, Hwang YK. Practical Path Planning Among Movable Obstacles. In: ICRA. 1991. p. 444–449.

[24] Nieuwenhuisen D, Frank van der Stappen A, Overmars MH. An Effective Framework for Path Planning amidst Movable Obstacles. In: WAFR. 2006.

[25] Botea A, Muller M, Schaffer J. Using Abstraction for Planning in Sokoban. Computers and Games. 2002;:360–375.

[26] Pereira AG, Ritt MRP, Buriol LS. Finding Optimal Solutions to Sokoban Using Instance Dependent Pattern Databases. In: Symposium on Combinatorial Search (SoCS). 2013.

[27] Reyes Castro LI, Chaudhari P, Tumova J, Karaman S, Frazzoli E, Rus D. Incremental sampling-based algorithm for minimum-violation motion planning. In: Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on. IEEE. 2013. p. 3217–3224.

[28] Dantam NT, Kingston ZK, Chaudhuri S, Kavraki LE. Incremental task and motion planning: a constraint-based approach. In: Proceedings of robotics: science and systems. 2016.

[29] Kaelbling LP, Lozano-Pérez T. Implicit belief-space pre-images for hierarchical planning and execu-

tion. In: Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE. 2016. p. 5455–5462.

[30] Kavraki LE, Svestka P, Latombe JC, Overmars M. Probabilistic Roadmaps for Path Planning in High-Dim. Configuration Spaces. IEEE TRA. 1996;.

[31] LaValle SM, Kuffner JJ. Randomized Kinodynamic Planning. IJRR. 2001;.

[32] Berenson D, Srinivasa SS, Kuffner JJ. Task Space Regions: A Framework for Pose-Constrained Manipulation Planning. IJRR. 2012;30(12):1435–1460.

[33] Alami R, Siméon T, Laumond JP. A Geometrical Approach to Planning Manipulation Tasks. In: ISRR. 1989. p. 113–119.