Article

The International Journal of Robotics Research
2018, Vol. 37(13-14) 1775–1795

DOI: 10.1177/0278364918780999
journals.sagepub.com/home/ijr

Complexity Results and Fast Methods for Optimal Tabletop Rearrangement with Overhand Grasps

Shuai D Han, Nicholas M Stiffler, Athanasios Krontiris, Kostas E Bekris and Jingjin Yu

Abstract

This paper studies the underlying combinatorial structure of a class of object rearrangement problems, which appear frequently in applications. The problems involve multiple, similar-geometry objects placed on a flat, horizontal surface, where a robot can approach them from above and perform pick-and-place operations to rearrange them. The paper considers both the case where the start and goal object poses overlap, and where they do not. For overlapping poses, the primary objective is to minimize the number of pick-and-place actions and then to minimize the distance traveled by the end-effector. For the non-overlapping case, the objective is solely to minimize the travel distance of the end-effector. Although such problems do not involve all the complexities of general rearrangement, they remain computationally hard in both cases. This is shown through reductions from well-understood, hard combinatorial challenges to these rearrangement problems. The reductions are also shown to hold in the reverse direction, which enables the convenient application on rearrangement of well-studied algorithms. These algorithms can be very efficient in practice despite the hardness results. The paper builds on these reduction results to propose an algorithmic pipeline for dealing with the rearrangement problems. Experimental evaluation, including hardware-based trials, shows that the proposed pipeline computes high-quality paths with regards to the optimization objectives. Furthermore, it exhibits highly desirable scalability as the number of objects increases in both the overlapping and non-overlapping setup.

Keywords

Object rearrangement, robot manipulation, path planning

1. Introduction

In many industrial and logistics applications, such as those shown in Figure 1, a robot is tasked to rearrange multiple, similar objects placed on a tabletop into a desired arrangement. In these setups, the robot needs to approach the objects from above and perform a *pick-and-place action* at desired target poses. Such operations are frequently part of product packaging and inspection processes. Efficiency plays a critical role in these domains, as the speed of task completion has a direct impact on financial viability; even a marginal improvement in productivity could provide significant competitive advantage in practice. Beyond industrial robotics, a home assistant robot may need to deal with such problems as part of a room-cleaning task. The reception of such a robot by people will be more positive if its solutions are efficient and the robot does not waste time performing redundant actions. Many subtasks affect the efficiency of the overall solution in all of these applications, ranging from perception to the robot's speed in grasping and transferring objects. However, overall efficiency also critically depends on the underlying combinatorial aspects of the problem, which relate to the number of pick-and-place actions that the robot performs, the placement of the objects, as well as the sequence of objects transferred.

This paper deals with the combinatorial aspects of tabletop object rearrangement tasks. The objective is to understand the underlying structure and obtain high-quality solutions in a computationally efficient manner. The focus is on a subset of general rearrangement problems, which relate to the above-mentioned applications. In particular, the setup corresponds to rearranging multiple, similar-geometry, non-stacked objects on a flat, horizontal surface from given start to goal arrangements. The robot can approach the objects from above, pick them up, and raise them. At that point, it can move them freely without collisions with other objects.

Computer Science Department, Rutgers, the State University of New Jersey, Piscataway, NJ, USA

Corresponding author:
Jingjin Yu, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854, USA.
Email: jingjin.yu@cs.rutgers.edu

Fig. 1. Examples of robots deployed in industrial settings tasked to arrange objects in desired configurations through pick and place: (left) ABB's IRB 360 FlexPicker rearranging pancakes; (right) Stäubli's TP80 Fast Picker robot.

This work mainly focuses on the case that the objects are labeled, which corresponds to an individual goal pose for each object. The less-restrictive unlabeled case, where objects can be moved to any one of potentially many shared goal poses, is also briefly discussed when differences in formulation arise.

There are two important variations of tabletop object rearrangement problem. The first requires that the target object poses do not overlap with the initial ones. In this scenario, the number of pick-and-place actions is equal to the number of objects not in their goal pose. Thus, the solution quality is dependent upon the sequence with which the objects are transferred. A good sequence can minimize the distance that the robot's end-effector travels. The second variant of the problem allows for goal poses to overlap with the start poses, as in Figure 2(a)–(c). The situation sometimes necessitates the identification of intermediate poses for some objects to complete the task. In such cases, the quality of the solution tends to be dominated by the number of pick-and-place actions the robot must carry out, which correlate with the usage of intermediate poses. The primary objective is to find a solution that uses the minimum number of pick-and-place actions and among them minimize the distance the robot's end-effector travels.

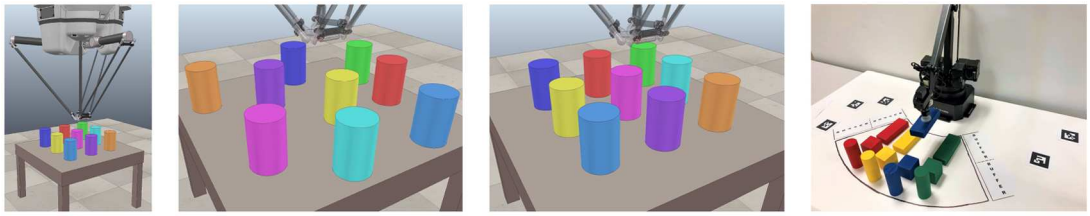
Both variations include some assumptions that simplify these instances relative to the general rearrangement problem. The non-overlapping case in particular seems to be quite easy since a random feasible solution can be trivially acquired. Nevertheless, this paper shows that even in this simpler setup, the optimal variant of the problem remains computationally hard. This is achieved by reducing the Euclidean-TSP problem (Papadimitriou, 1977) to the cost-optimal, non-overlapping tabletop object rearrangement problem. Even in the unlabeled case, the problem is still hard. For overlapping start and goal poses, the paper employs a graphical representation from the literature (van den Berg et al., 2009), which leads to the result that finding the minimum number of pick-and-place actions relates to a well-known problem in the algorithmic community, the "feedback vertex set" (FVS) problem (Karp, 1972). This again indicates the hardness of the challenge.

The benefit of these two-way reductions, beyond the hardness results themselves, is that they suggest algorithmic solutions and provide an expectation on the practical efficiency of the methods. On the one hand, Euclidean-TSP admits a polynomial-time approximation scheme (PTAS) and good heuristics, which implies very good practical solutions for the non-overlapping case. Indeed, effective solvers have been developed (e.g., the Concorde TSP solvers (Applegate et al., 2007)), which can compute optimal solutions to instances with thousands of vertices in seconds to minutes. On the other hand, the FVS problem is APX-hard (Dinur and Safra, 2005; Karp, 1972), which indicates that efficient algorithms are harder for the overlapping case. For the relatively simpler undirected case, the best known approximate algorithm for FVS incurs an error factor of two (Bafna et al., 1999; Monien and Speckenmeyer, 1985), which can be unsatisfactory for practical applications. The directed case (Even et al., 1998) is much harder and is difficult to even approximate. This motivated the consideration of alternative heuristics for solving such challenges that make sense in the context of object rearrangement.

The algorithms proposed here, which arose by mapping the object rearrangement variations to well-studied problems, have been evaluated in terms of practical performance. For the non-overlapping case, an alternative solver exists that was developed for a related challenge (Treleaven et al., 2013). The TSP solvers achieve superior performance relative to this alternative when applied to object rearrangement. They achieve sub-second solution times for hundreds of objects. Optimal solutions are shown to be significantly better than the average, random, feasible solution. For the overlapping case, exact and heuristic solvers are considered. The paper shows that practically efficient methods achieve sub-second solution times without a major impact in solution quality for tens of objects. To the best of the authors' knowledge, this is the first work that takes full advantage of the *dependency graph* as a formal tool in the complexity and algorithmic study of the tabletop object rearrangement problem. This approach not only allows the establishment of the computational intractability of such problems but also provides a route to obtaining high-quality solutions.

This article expands an earlier version of this work (Han et al., 2017) and includes the following new contributions: (i) a proof showing that cost-optimal unlabelled non-overlapping tabletop rearrangement is NP-hard; (ii) reductions from tabletop object rearrangement problems to Euclidean-TSP and FVS; (iii) a complete description of ILP-based heuristics and an algorithm for solving the object rearrangement problem with overlap sub-optimally; (iv) significantly enhanced evaluation with experiments conducted on a hardware platform (Figure 2(d)), corroborating the real-world benefits of the proposed method.

The structure of this manuscript is as follows. Section 2 reviews related work, followed by a formal problem statement in Section 3. Section 4 considers the object rearrangement problem when the objects have non-overlapping



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

Fig. 2. (a)–(c) An example of an object rearrangement challenge considered in this work from a V-REP simulation (Rohmer et al., 2013), where the initial (b) and final (c) object poses are overlapping and an object needs to be placed at an intermediate location. (d) In addition to the V-REP simulations, the solutions presented in this work have been tested on an experimental hardware platform.

start and goal arrangements. The more computationally challenging problem that involves overlapping start and goal arrangements appears in Section 5. The evaluation of the proposed methods is broken down into two components: simulation (Section 6) and physical experiments (Section 7). The paper concludes with a summary and remarks about future directions in Section 8.

2. Contribution relative to prior work

Multi-body planning is a related challenge that is itself hard. In the general, continuous case, complete approaches do not scale even though methods exist that try to decrease the effective degrees of freedom (DOFs) (Aronov et al., 1999). For specific geometric setups, such as unlabeled unit disks among polygonal obstacles, optimality can be achieved (Solovey et al., 2015), even though the unlabeled case is still hard (Solovey and Halperin, 2015). Given the hardness of multi-robot planning, decoupled methods, such as priority-based schemes (van den Berg and Overmars, 2005) or velocity tuning (Leroy et al., 1999), trade completeness for efficiency. Assembly planning (Halperin et al., 2000; Sundaram et al., 2001; Wilson and Latombe, 1994) deals with similar problems but few optimality arguments have been made.

Recent progress has been achieved for the discrete variant of the problem, where robots occupy vertices and move along edges of a graph. For this problem, also known as "pebble motion on a graph" (Auletta et al., 1999; Calinescu et al., 2008; Goraly and Hassin, 2010; Kornhauser et al., 1984), feasibility can be answered in linear time and paths can be acquired in polynomial time. The optimal variation is still hard but recent optimal solvers with good practical efficiency have been developed either by extending heuristic search to the multi-robot case (Sharon et al., 2015; Wagner et al., 2012), or utilizing solvers for other hard problems, such as network flow (Yu and LaValle, 2012, 2016). The current work is motivated by this progress and aims to show that for certain useful rearrangement setups it is possible to come up with practically efficient algorithms through an understanding of the problem's structure.

Navigation among movable obstacles (NAMO) is a related computationally hard problem (Chen and Hwang, 1991; Demaine et al., 2000; Nieuwenhuisen et al., 2006; Wilfong, 1991), where a robot moves and pushes objects. A probabilistically complete solution exists for this problem (van den Berg et al., 2008). NAMO can be extended to manipulation among movable obstacles (MAMO) (Stilman et al., 2007) and rearrangement planning (Ben-Shahar and Rivlin, 1998; Ota, 2004). Monotone instances for such problems, where each obstacle may be moved at most once, are easier (Stilman et al., 2007). Recent work has focused on "non-monotone" instances (Garrett et al., 2014; Havur et al., 2014; Krontiris and Bekris, 2015, 2016; Krontiris et al., 2014; Srivastava et al., 2014). Rearrangement with overlaps considered in the current paper includes "non-monotone" instances although other aspects of the problem are relaxed. In all these efforts, whereas sometimes certain optimization techniques are used to facilitate the computation, the focus is mainly on feasibility and no solution quality arguments have been provided. In contrast, this study focuses on the optimality aspects of a (limited) class of object rearrangement problems, incorporating optimality considerations deeply into the problem formulation. Asymptotic optimality has been achieved for the related "minimum constraint removal" path problem (Hauser, 2014), which, however, does not consider negative object interactions.

The *pickup and delivery problem* (PDP) (Berbeglia et al., 2007, 2010) is a well-studied problem in operations research that is similar to tabletop object rearrangement, as long as the object geometries are ignored. The PDP models the pickup and delivery of goods between different parties and can be viewed as a subclass of vehicle routing (Laporte, 1992) or dispatching (Christofides and Eilon, 1969). It is frequently specified over a graph embedded in the 2D plane, where a subset of the vertices are pickup and delivery locations. A PDP in which pickup and delivery sites are not uniquely paired is also known as the NP-hard swap problem (Anily and Hassin, 1992; Garey and Johnson, 1979), for which a 2.5-optimal heuristic is known (Anily and Hassin, 1992). Many exact linear programming algorithms and approximations are available (Beullens et al., 2004; Gribkovskaia et al., 2007; Hoff and Løkketangen, 2006) when pickup and delivery locations overlap, where pickup must happen some time after delivery. The stacker crane problem (SCP) (Frederickson et al., 1976; Treleaven

et al., 2013) is a variation of PDP of particular relevance as it maps to the non-overlapping case of labeled object rearrangement. An asymptotically optimal solution for SCP (Treleaven et al., 2013) is used as a comparison point in the evaluation section.

This work does not deal with other aspects of rearrangement, such as arm motion (Berenson et al., 2012; Cohen et al., 2013; Siméon et al., 2004; Zucker et al., 2013) or grasp planning (Bohg et al., 2014; Ciocarlie and Allen, 2009). Non-prehensile actions, such as pushing, are also not considered (Cosgun et al., 2011; Dogar and Srinivasa, 2011). Similar combinatorial issues to those studied here are also studied by integrated task and motion planners, for most of which there are no optimality guarantees (Cambon et al., 2009; Dantam et al., 2016; Garrett et al., 2014; Gharbi et al., 2015; Plaku and Hager, 2010; Srivastava et al., 2014). Recent work on asymptotically optimal task planning is at this point prohibitively expensive for practical use (Vega-Brown and Roy, 2016).

3. Problem statement

This section formally defines the considered challenges.

3.1. Tabletop object rearrangement with overhand grasps

Consider a workspace $\mathcal{W}$ with static obstacles and a set of n movable objects $\mathcal{O} = \{o_1, \ldots, o_n\}$. For $o_i \in \mathcal{O}$, $\mathcal{C}_i$ denotes its configuration space. Then, $\mathcal{F}_i \subseteq \mathcal{C}_i$ is the set of collision-free configurations of o_i with respect to the static obstacles in $\mathcal{W}$. An *arrangement* $R = (r_1, \ldots, r_n)$ for the objects $\mathcal{O}$ specifies a configuration $r_i \in \mathcal{F}_i$ for each object o_i. To avoid collisions between objects, R must satisfy that for all $1 \leq i < j \leq n$, r_i and r_j are not in collision.

This work focuses on closed and bounded planar workspaces: $\mathcal{W} \subset \mathbb{R}^2$. The setting is frequently referred to as the *tabletop* setup, in which the vertical projections of the objects on the tabletop do not intersect. In this case, $\mathcal{C}_i = SE(2)$ for all $1 \leq i \leq n$. This work assumes that the manipulator is able to employ *overhand* grasps, where an object can be transferred after being lifted above all other objects. In particular, a pick-and-place operation of the manipulator involves four steps:

(a) bringing the end-effector above the object;
(b) grasping and lifting the object;
(c) transfer of the grasped object horizontally to its target (horizontal) location; and
(d) a downward motion prior to releasing the object.

This sequence constitutes a *pick-and-place action*, which is denoted as a.

The manipulator is initially at a rest position s_M prior to executing any pick-and-place actions, and transitions to a rest position g_M at the conclusion of the rearrangement task. A *rest position* is a safe arm configuration, where there is no collision with objects.

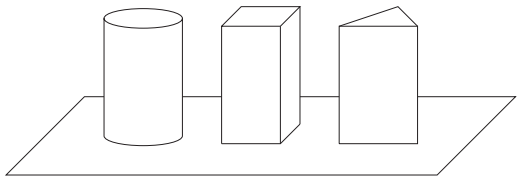


Fig. 3. Examples of general cylinders placed on a 2D plane.

The illustrations that appear throughout the paper assume objects are of identical geometry. Nevertheless, the results derived in this paper are not dependent on this assumption, i.e. objects need only be general cylinders (in other words, *cylindrical*). In geometry, a *general cylinder* is defined as any ruled surface spanned by a one-parameter family of parallel lines. Note that under this definition, the footprint of an object on the tabletop can be an arbitrary polygon, instead of a circle. Figure 3 demonstrates several examples of *general cylinders*.

Given the setup, the problem studied in the paper can be summarized as follows.

Problem 1 (Tabletop Object Rearrangement with Overhand grasps (`TORO`))**.** *Given start and goal arrangements R_S, R_G for objects $\mathcal{O} = \{o_1, \ldots, o_n\}$ on a tabletop, determine a finite sequence of collision-free pick-and-place actions $\mathcal{A} = (a^1, a^2, \ldots)$ that transfer $\mathcal{O}$ from R_S to R_G.*

A rearrangement problem is said to be *labeled* if objects are unique and not interchangeable. Otherwise, the problem is *unlabeled*. If for two arbitrary arrangements $s \in R_S$ and $g \in R_G$, the objects placed in s and g are not in collision, then the problem is said to have *no overlaps*. Otherwise, the problem is said to have *overlaps*.

This paper primarily focuses on the labeled `TORO` case and identifies an important subcase:

- *`TORO` with NO overlaps (`TORO-NO`)*

Remark 1. The partition of Problem 1 into the general `TORO` case and the subcase of `TORO-NO` is not arbitrary. `TORO` is structurally richer and harder from a computational perspective. Both versions of the problem can be extended to the unlabeled and partially labeled variants. This paper does not treat the labeled and unlabeled variants as separate cases but will briefly discuss differences that arise due to formulation when appropriate.

3.2. Optimization criteria

Recall that a pick-and-place action a^i has four components: an initial move, a grasp, a transport phase, and a release. As grasping is frequently the source of difficulty in object manipulation tasks, it is assumed in the paper that grasps and subsequent releases induce the most cost in pick-and-place actions. The other source of cost can be attributed to the length of the manipulator's path. This part of the cost is captured through the Euclidean distance traveled by the

end-effector between grasps and releases. For a pick-and-place action a^i, the incurred cost is

$$c_{a^i} = c_m d_e^i + c_g + c_m d_l^i + c_r \tag{1}$$

where c_m, c_g, c_r are costs associated with moving the manipulator, a single grasp, and a single release, respectively. d_e^i and d_l^i are the straight line distances traveled by the end-effector in the first (object-free) and third (carrying an object) stages of a pick-and-place action, respectively.

The total cost associated with solving a TORO instance is then captured by

$$c_T = \sum_{i=1}^{|\mathcal{A}|} c_{a^i} = |\mathcal{A}|(c_g + c_r) + c_m\Big(\sum_{i=1}^{|\mathcal{A}|}(d_e^i + d_l^i) + d_f\Big) \tag{2}$$

where d_f is the distance between the location of the last release of the end-effector and its rest position g_M. Of the two additive terms in (2), note that the first term dominates the second. Because the absolute value of c_g, c_r, and c_m are different for different systems, the assignment of their absolute values is left to practitioners. The focus of this paper is the analysis and minimization of the two additive terms in (2).

3.3. Object buffer locations

The resolution of TORO (Section 5) may require the temporary placement of some object(s) at intermediate locations outside those in $R_S \cup R_G$. When this occurs, external buffer locations may be used as temporary locations for object placement. More formally, there exists a set of configurations, called *buffers*, which are available to the manipulator and do not overlap with object placements in R_S or R_G.

Remark 2. This work, which focuses on the combinatorial aspects of multi-object manipulation and rearrangement, utilizes exclusively buffers that are not on the tabletop. It is understood that the number of external buffers *may* be reduced by attempting to first search for potential buffers within the tabletop. Nevertheless, there are scenarios where the use of external buffers may be necessary.

3.4. Summary and discussion of assumptions

The main assumptions are listed as follows:

(A1) the workspace is a bounded 2D region (e.g. *tabletop*);
(A2) objects are *general cylinders* and with identical geometry;
(A3) the poses of the objects are perfectly known, so as to be able to perform accurate collision checking;
(A4) only a single manipulator is considered;
(A5) the manipulator performs overhand grasps;
(A6) there is a separate 2D region with unlimited space to use as a buffer location;
(A7) the optimization objective is to first minimize the number of pick-and-place actions and then minimize the total (horizontal) travel distance of the end-effector.

Note: these assumptions are identified to precisely specify the scope of this work. The results from this study provide general insights that can apply beyond the constraints imposed by many of these assumptions.

4. TORO with no overlaps (TORO-NO)

When there is no overlap between any pair of start and goal configurations, an object can be transferred directly from its start configuration to its goal configuration. A direct implication is that an optimal sequence of pick-and-place actions contains exactly $|\mathcal{A}| = |\mathcal{O}| = n$ grasps and the same number of releases. Note that a minimum of n grasps and releases are necessary. This also implies that no buffer is required because using buffers will incur additional grasp and release costs. Therefore, for TORO-NO, (2) becomes

$$c_T = n(c_g + c_r) + c_m\Big(\sum_{i=1}^{n}(d_e^i + d_l^i) + d_f\Big), \tag{3}$$

i.e. only the distance traveled by the end-effector affects the cost. The problem instance that minimizes (3) is referred to as Cost-optimal TORO-NO. The following theorem provides a hardness result for Cost-optimal TORO-NO.

Theorem 4.1. *Cost-optimal TORO-NO is NP-hard.*

Proof. Reduction from Euclidean-TSP (Papadimitriou, 1977). Let $p_0, p_1, \ldots, p_n$ be an arbitrary set of $n+1$ points in two dimensions. The set of points induces an Euclidean-TSP. Let d_{ij} denote the Euclidean distance between p_i and p_j for $0 \le i,j \le n$. In the formulation given in Papadimitriou (1977), it is assumed that d_{ij} are integers, which is equivalent to assuming the distances are rational numbers. To reduce the stated TSP problem to a cost-optimal TORO-NO problem, pick some positive $\varepsilon \ll 1/(4n)$. Let p_0 be the rest position of the manipulator in an object rearrangement problem. Denoting $s_i = (s_{i1}, s_{i2})$ and $g_i = (g_{i1}, g_{i2})$ as the 2D positions (coordinates) of objects, for each p_i, $1 \le i \le n$, split p_i into a pair of start and goal configurations (s_i, g_i) such that (i) $p_i = \frac{s_i + g_i}{2}$, (ii) $s_{i2} = g_{i2}$, and (iii) $s_{i1} + \varepsilon = g_{i1}$. An illustration of the reduction is provided in Figure 4. The reduced TORO-NO instance is fully defined by p_0, $R_S = \{s_1, \ldots, s_n\}$ and $R_G = \{g_1, \ldots, g_n\}$. Intuitively, the start and goal configurations of each object in this TORO-NO instance are very close to each other. A cost-optimal (as defined by (3)) solution to this TORO-NO problem induces a (closed) path starting from p_0, going through each s_i and g_i exactly once, and ending at p_0. Moreover, each g_i is visited immediately after the corresponding s_i is visited. Based on this path, the manipulator moves to a start location to pick up an object, drop the object at the corresponding goal configuration, and then move to the next

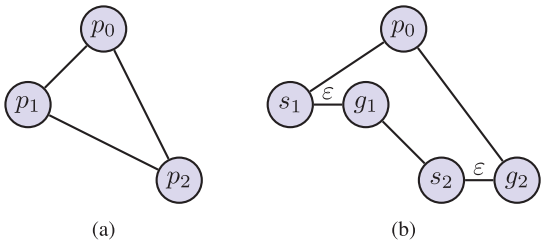


Fig. 4. Reduction from Euclidean-TSP to cost-optimal TORO-NO: (a) the optimal solution path P_{opt} of the Euclidean-TSP, where $\{p_0, p_1, p_2\}$ is the set of points in two dimensions; (b) the cost-optimal solution of the reduced TORO-NO. Here, p_0 is the initial and rest pose of the manipulator (s_M, g_M) and s_1, g_1 (respectively s_2, g_2) are the start and goal configurations of object o_1 (respectively o_2).

object until all objects are rearranged. Denote the loop path as P and let its total length be D.

Assume that the Euclidean-TSP has an optimal solution path P_{opt} with a total distance of D_{opt} (an integer). Further assume that this solution is unique. For example, if this is not the case, it is possible to simply duplicate each p_i multiple times to force this; this does not impact the reduction proof strategy. The first step is to establish that this solution P_{opt} induces an optimal solution to the TORO-NO problem. The solution itself is obtained directly through the reduction process: without loss of generality, suppose the optimal TSP tour goes through $p_0, p_1, \ldots p_n$. Then, the corresponding TORO-NO solution would be the end-effector trajectory P that goes through $p_0, s_1, g_1, s_2, g_2, \ldots, s_n, g_n$, in that order. Let the total distance of P be D, which is no more than $D_{opt} + n * 1/(4n) = D_{opt} + 1/4$ (via straightforward computation using the assumption that the distance between any p_i and p_j is some positive integer). If P is not an optimal solution to the TORO-NO instance, there is another better solution P' with total length D'. Nevertheless, from P', if all s_i, g_i pairs are contracted to the same point, the resulting path is a feasible solution to the original TSP problem with a total length of no less than $D' - n * 1/(4n) = D' - 1/4 \geq D_{opt} + 1 > D$, a contradiction.

For the reverse direction, the claim is that the solution P of a cost-optimal TORO-NO yields an optimal path for the TSP, i.e. P_{opt} with total distance D_{opt}. To show this, from P, we again contract the edges $s_i g_i$ for all $1 \leq i \leq n$. This clearly yields a solution to the Euclidean-TSP; let the resulting path be P' with total length D'. As edges are contracted along P, by the triangle inequality, $D' \leq D$. It remains to show that $D' = D_{opt}$. Suppose this is not the case, then $D' \geq D_{opt} + 1$. However, if this is the case, a solution to the TORO-NO can be constructed by splitting p_i into s_i and g_i along P_{opt}. It is straightforward to establish that the total distance of this TORO-NO path is bounded by $D_{opt} + n\varepsilon < D_{opt} + n * 1/(4n) = D_{opt} + 1/4 < D_{opt} + 1 \leq D' \leq D$. As this is a contradiction, $D' = D_{opt}$. □

Algorithm 1: TORONOTSP

Input: Configurations s_M, g_M, Arrangements R_S, R_G.
Output: A sequence of pick-and-place actions $\mathcal{A}$.

1 $G_{NO} \leftarrow$ CONSTRUCTTSPGRAPH(R_S, R_G, s_M, s_G)
2 $S_{raw} \leftarrow$ SOLVETSP(G_{NO})
3 $\mathcal{A} \leftarrow$ RETRIEVEACTIONS(S_{raw})
4 **return** $\mathcal{A}$

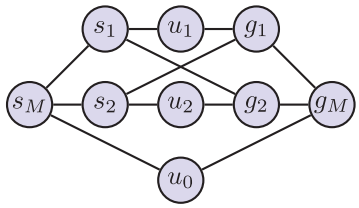


Fig. 5. An example of G_{NO} when $n = 2$. The possible Hamiltonian cycles of this graph are: $(s_M, s_1, u_1, g_1, s_2, u_2, g_2, g_M, u_0)$, corresponds to a TORO-NO solution which moves o_1 and then o_2; and $(s_M, s_2, u_2, g_2, s_1, u_1, g_1, g_M, u_0)$, corresponds to a TORO-NO solution which moves o_2 and then o_1.

Remark 3. Note that an NP-hardness proof of a similar problem can be found in Frederickson and Guan (1993), as is mentioned in Treleaven et al. (2013). Nevertheless, the problem is stated for a tree and is non-Euclidean. Furthermore, it is straightforward to show that the decision version of the cost-optimal TORO-NO problem is NP-complete; the detail, which is non-essential to the focus of the current paper, is omitted.

Remark 4. Interestingly, TORO-NO may also be reduced to a variant of TSP with very little overhead. Because highly efficient TSP solvers are available, the reduction route provides an effective approach for solving TORO-NO. That is, a TORO-NO instance may be reduced to TSP and solved, with the TSP solution readily translated back to a solution to the TORO-NO instance that is cost-optimal. This is not always a feature of NP-hardness reductions. The straightforward algorithm for the computation is outlined in Algorithm 1. The inputs to the algorithm are the rest positions of the manipulator and the start and goal arrangements of objects. The output is the solution for TORO-NO, represented as a sequence of pick-and-place actions $\mathcal{A}$.

In line 1 of Algorithm 1, a graph $G_{NO}(V_{NO}, E_{NO})$ is generated as the input to the TSP problem. The graph is constructed from the TORO-NO instance as follows. A vertex is created for each element of R_S and R_G. Then, a complete bipartite graph is created between these two sets of vertices. A set of vertices $U = \{u_1, \ldots, u_n\}$ is then inserted into edges $s_i g_i$ for $1 \leq i \leq n$. Afterward, s_M (respectively, g_M) is added as a vertex and is connected to s_i (respectively, g_i) for $1 \leq i \leq n$. Finally, a vertex u_0 is added and connected to both s_M and g_M. See Figure 5 for the straightforward example for $n = 2$.

Let $w(a,b)$ denote the weight of an edge $(a,b) \in E_{NO}$. For all $1 \leq i,j \leq n, i \neq j$ ($\text{dist}(x,y)$ denotes the Euclidean distance between x and y in two dimensions):

$$w(s_M,u_0) = w(g_M,u_0) = 0, \quad w(s_M,s_i) = \text{dist}(s_M,s_i)$$
$$w(g_M,g_i) = \text{dist}(g_M,g_i), \quad w(s_i,u_i) = w(u_i,g_i) = 0$$
$$w(s_i,g_j) = \text{dist}(s_i,g_j)$$

With the construction, a TSP tour through G_{NO} must use $s_M u_0 g_M$ and all $s_i u_i g_i$ for all $1 \leq i \leq n$. To form a complete tour, exactly $n-1$ edges of the form $g_i s_j$, where $i \neq j$ must be used. In line 2, the TSP is solved (using Concorde TSP solver (Applegate et al., 2007)). This yields a minimum weight solution S_{raw}, which is a cycle containing all $v \in V_{NO}$. The pick-and-place actions can then be retrieved (line 3).

An alternative solution to TORO-NO is to employ the SPLICE algorithm, introduced in Treleaven et al. (2013), which is designed for pickup and delivery problems. The SPLICE algorithm involves two steps. First, compute the optimal bipartite matching, which defines the next object to be picked up after the placement of each object. This procedure produces, with high probability, an infeasible solution with multiple sub-tours. The next step is to connect the sub-tours in a greedy manner. Given the fact that the number of sub-tours is $O(\log n)$ in expectation, the algorithm is proven to be asymptotically optimal in terms of the number of objects. Its computational time is $O(n^{2+\varepsilon})$, where ε is a small positive real number. Section 6 provides a comparison between ToroNoTSP and SPLICE based on computational time and optimality.

4.1. Unlabeled TORO-NO

The scenario where objects are unlabeled is a special case of TORO-NO which has significance in real-world applications (e.g., the pancake stacking application). This case is denoted as TORO-UNO (unlabeled, no overlap). Adapting the NP-hardness proof for the TORO-NO problem shows that cost-optimal TORO-UNO is also NP-hard. Similar to the TORO-NO case, the optimal solution only hinges on the distance traveled by the manipulator because no buffer is required and exactly n grasps and releases are needed.

Theorem 4.2. *Cost-optimal TORO-UNO is NP-hard.*

Proof. See Appendix B. □

When solving a TORO-UNO instance, Algorithm 1 may be used with a few small changes. First, a different underlying graph must be constructed. Denote the new graph as $G_{UNO}(V_{UNO}, E_{UNO})$, where $V_{UNO} = R_S \cup R_G \cup \{s_M, u_0, g_M\}$. For all $1 \leq i,j \leq n$:

$$w(s_M,u_0) = w(g_M,u_0) = 0, w(s_M,s_i) = \text{dist}(s_M,s_i)$$
$$w(g_M,g_i) = \text{dist}(g_M,g_i), w(s_i,g_j) = \text{dist}(s_i,g_j)$$

All other edges are given infinite weight. An example of the updated structure of G_{UNO} for two objects is illustrated in Figure 6.

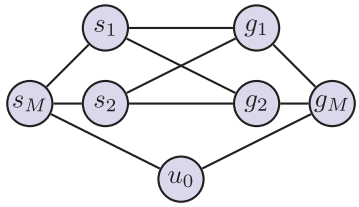


Fig. 6. An example of G_{UNO} when $n = 2$. The possible Hamiltonian cycles of this graph are: $(s_M,s_1,g_1,s_2,g_2,g_M,u_0)$, $(s_M,s_1,g_2,s_2,g_1,g_M,u_0)$, $(s_M,s_2,g_1,s_1,g_2,g_M,u_0)$, and $(s_M,s_2,g_2,s_1,g_1,g_M,u_0)$. Each Hamiltonian cycle corresponds to a feasible solution of the original TORO-UNO.

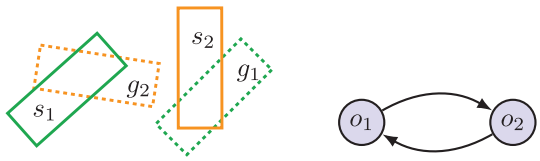


Fig. 7. Illustration of the dependency graph. (a) Two objects are to be moved from s_i to g_i, $i = 1,2$. Owing to the overlap between s_1 and g_2 as well as the overlap between s_2 and g_1, one of the objects must be temporarily moved aside. (b) The dependency graph capturing the scenario in (a).

5. TORO with overlap (TORO)

Unlike TORO-NO, TORO has a more sophisticated structure and may require buffers to solve. In this section, a *dependency graph* (van den Berg et al., 2009) is used to model the structure of TORO, which leads to a classical NP-hard problem known as the FVS problem (Karp, 1972). The connection then leads to a complete algorithm for optimally solving TORO.

5.1. The dependency graph and NP-hardness of TORO

Consider a *dependency digraph* $G_{dep}(V_{dep}, A_{dep})$, where $V_{dep} = \mathcal{O}$, and $(o_i,o_j) \in A_{dep}$ iff g_i and s_j overlap. Therefore, o_j must be moved away from s_j before moving o_i to g_i. An example involving two objects is provided in Figure 7. The definition of dependency graph implies the following two observations.

Observation 5.1. *If the out-degree of $o_i \in V_{dep}$ is zero, then o_i can move to g_i without collision.*

A zero out-degree of o_i indicates that there is no object blocking its goal configuration.

Observation 5.2. *If G_{dep} is not acyclic, solving TORO requires at least $n+1$ grasps.*

When G_{dep} has cycles, pick an arbitrary cycle in G_{dep} with $n_c \leq n$ objects. Then, in this cycle, because all the objects have positive out-degree, none of these objects can be directly moved to their goal configuration. In this case,

at least one object needs to be moved to a buffer, and to be moved to its goal configuration afterwards. Thus, the minimum number of grasps required to solve the problem is at least $(n_c + 1) + (n - n_c) = n + 1$.

The dependency graph has obvious similarities to the well-known FVS problem (Karp, 1972). A directed FVS problem is defined as follows. Given a strongly connected directed graph $G = (V, A)$, an FVS is a set of vertices whose removal leaves G acyclic. Minimizing the cardinality of this set is NP-hard, even when the maximum in-degree or out-degree is no more than two (Garey and Johnson, 1979). As it turns out, the set of removed vertices in an FVS problem mirrors the set of objects that must be moved to temporary locations (i.e. buffers) for resolving the dependencies between the objects, which corresponds to the additional grasps (and releases) that must be performed in addition to the n required grasps for rearranging n objects. The observation establishes that cost-optimal TORO is also computationally intractable. The following lemma shows this point.

Lemma 5.1. *Let the dependency graph of a TORO problem be a single strongly connected graph. Then the minimum number of additional grasps (that exceeds n) required for solving the TORO problem equals the cardinality of the minimum FVS of the dependency graph.*

Proof. Given the dependency graph, let the additional grasps and releases be n_x and the minimum FVS have a cardinality of n_{fvs}, it remains to show that $n_x = n_{fvs}$. First, if fewer than n_{fvs} objects are removed, which correspond to vertices of the dependency graph, then there remains a directed cycle. By Observation 5.2, this part of the problem cannot be solved. This establishes that $n_x \geq n_{fvs}$. On the other hand, once all objects corresponding to vertices in a minimum FVS are moved to buffer locations, the dependency graph becomes acyclic. This allows the remaining objects to be rearranged. This operation can be carried out iteratively with objects whose corresponding vertices have no incoming edges. On a directed acyclic graph (DAG), there is always such a vertex. Moreover, as such a vertex is removed from a DAG, the remaining graph must still be a DAG and therefore must have either no vertex (a trivial DAG) or a vertex without incoming edges. □

For dependency graphs with multiple strongly connected components (SCCs), the required number of additional grasps and releases is simply the sum of the required number of such actions for the individual SCCs.

For a fixed TORO problem, let n_{fvs} be the cardinality of the largest (minimal) FVS computed over all SCCs of its dependency graph. Then it is easy to see that the maximum number of required buffers is no more than n_{fvs}. The NP-hardness of cost-optimal TORO is established using the reduction from FVS problems to TORO. This is more involved than reducing TORO to FVS because the constructed TORO must correspond to an actual TORO problem in which the number of overlaps should not grow unbounded.

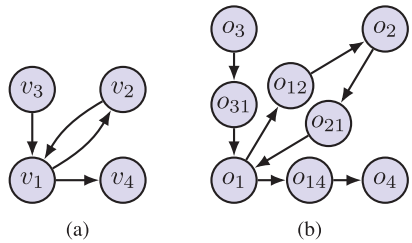


Fig. 8. Converting (a) a neighborhood of a graph for an FVS problem into (b) parts of a dependency graph for a TORO problem.

Theorem 5.1. *Cost-optimal TORO is NP-hard.*

Proof. The FVS problem on directed graphs is reduced to cost-optimal TORO. An FVS problem is fully defined by specifying an arbitrary strongly connected directed graph $G = (V, A)$ where each vertex has no more than two incoming and two outgoing edges. A typical vertex neighborhood can be represented as illustrated in Figure 8(a). Such a neighborhood is converted into a dependency graph neighborhood of object rearrangement as follows. Each of the original vertices $v_i \in V$ becomes an object o_i that has some (s_i, g_i) pair as its start and goal configurations. For each directed arc $v_i v_j$, split it into two arcs and add an additional object o_{ij}. That is, create new arcs $o_i o_{ij}$ and $o_{ij} o_j$ for each original arc v_{ij} (see Figure 8(b)). This yields a dependency graph that is again strongly connected. Two claims will be proven: □

1. the constructed dependency graph corresponds to an object rearrangement problem; and
2. the minimum number of objects that must be moved away temporarily to solve the problem is the same as the size of the minimum FVS.

To prove the first claim, assume without loss of generality that the objects have the same footprints on the tabletop. Furthermore, only the neighborhood of o_1 needs to be inspected because it is isolated by the newly added objects. Recall that an incoming edge to o_1 means that the start configuration o_1 blocks the goals of some other objects, in this case o_{21} and o_{31}. This can be readily realized by putting the goal configurations of o_{21} and o_{31} close to each other and have them overlap with the start configuration of o_1. Note that the goal configurations of o_{21} and o_{31} have no other interactions. Therefore, such an arrangement is always achievable for even simple (e.g. circular or square) footprints. Similarly, for the outgoing edges from o_1, which mean other objects block o_1's goal, in this case o_{12} and o_{14}, place the start configurations of o_{12} and o_{14} close to each other and make both overlap with the goal configuration of o_1. Again, the start configurations of o_{12} and o_{14} have no other interactions.

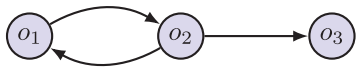


Fig. 9. An example of a dependency graph when $n = 3$.

The second claim follows Lemma 5.1 directly. Now, given an optimal solution to the reduced TORO problem, it remains to show that the solution can be converted into a solution to the original FVS problem. The solution to the TORO problem provides a set of objects that are moved to temporary locations. This yields a minimum FVS on the dependency graph but not the original graph G. Note that if a newly created object (e.g., o_{ij}) is moved to a temporary place, either object o_i or o_j can be moved as this will achieve no less in disconnecting the dependency graph. Doing this across the dependency graph yields a minimum FVS for G. That is, a small set of objects whose removal makes the residual dependency graph acyclic.

Remark 5. It is possible to prove that TORO is NP-hard using a similar proof to the TORO-NO case. To make the proof for Theorem 4.1 work here, each p_i can be split into an overlapping pair of start and goal. Such a proof, however, would bury the true complexity of TORO, which is a much more difficult problem. Unlike the Euclidean-TSP problem, which admits $(1+\varepsilon)$-approximations and good heuristics, FVS problems are APX-hard (Dinur and Safra, 2005; Karp, 1972).

5.2. Algorithmic solutions for TORO

From the discussion in Section 5.1, a subset of TORO is exactly FVS: when minimizing the number of grasps, to find the FVS set in the dependency graph is an FVS; however, an FVS set over the dependency graph does not necessarily map to a unique solution of the original TORO. Instead, it provides a set of objects whose removal make the residual dependency graph acyclic. As an example, for the dependency graph shown in Figure 9, a possible FVS set could be $\{o_2\}$, while the solutions that move o_2 to a buffer is multiple. It can possibly be (i) move o_3 to g_3, (ii) move o_2 to a buffer, (iii) move o_1 to g_1, (iv) move o_2 to g_2, or (i) move o_2 to a buffer, (ii) move o_3 to g_3, (iii) move o_1 to g_1, (iv) move o_2 to g_2.

5.2.1. A complete algorithm A complete solution of TORO then targets at minimizing the travel distance of the end-effector based on the FVS solution. Such a complete algorithm is outlined in TOROFVSSINGLE (Algorithm 2). In line 1, the dependency graph G_{dep} is constructed. In lines 3–4, an FVS is obtained for each SCC in G_{dep}. Note that if these FVSs are optimal, then it yields the minimum number of required grasps (and releases) as $\min |\mathcal{A}| = n + |B|$.

The residual work is to find the solution with $n + |B|$ grasps and the shortest travel distance (line 5). In line 6, the pick-and-place actions are then retrieved and returned.

Algorithm 2: TOROFVSSINGLE

Input: Configurations s_M, g_M, Arrangements R_S, R_G
Output: A set of pick-and-place actions $\mathcal{A}$

1 $G_{dep} \leftarrow$ CONSTRUCTDEPGRAPH(R_S, R_G)
2 $B \leftarrow \emptyset$
3 **for** *each SCC* ***in*** G_{dep} **do**
4 $\quad B \leftarrow B \cup$ SOLVEFVS(SCC)
5 $S_{raw} \leftarrow$ MINDIST($s_M, g_M, R_S, R_G, G_{dep}, B$)
6 $\mathcal{A} \leftarrow$ RETRIEVEACTIONS(S_{raw})
7 **return** $\mathcal{A}$

This paper explores two exact and three approximate methods as implementations of SOLVEFVS() (line 4 of Algorithm 2). The two exact methods are both based on integer linear programming (ILP) models, similar to those introduced in Baharev et al. (2015). They differ in how cycle constraints are encoded: one uses a polynomial number of constraints and the other simply enumerates all possible cycles. Denote these two exact methods as **ILP-Constraint** and **ILP-Enumerate**, respectively. The details of these two exact methods are explained in Appendix C. With regards to approximate solutions, several heuristic solutions are presented.

1. **Maximum simple cycle heuristic (MSCH)**. The FVS is obtained by iteratively removing the node that appears on the most number of simple cycles in G_{dep} until no more cycles exist. The simple cycles are enumerated.
2. **Maximum cycle heuristic (MCH)**. This heuristic is similar to MSCH, but counts cycles differently. For each vertex $v \in V_{dep}$, it finds a cycle going through v and marks the outgoing edge from v on this cycle. The process is repeated for v until no more cycles can be found. The vertex with the largest cycle count is then removed first.
3. **Maximum degree heuristics (MDH)**. This heuristic constructs an FVS through vertex deletion based on the degree of the vertex until no cycles exist.

Based on FVS, the solution minimizing travel distance can be found by MINDIST() (line 5), which is an ILP modeling method inspired by (Yu and LaValle, 2016) and described in Appendix D.

5.2.2. Global optimality Note that TOROFVSSINGLE() is a complete algorithm for solving TORO but it is not a complete algorithm for solving TORO optimally. With some additional engineering, a complete optimal TORO solver can also be constructed: under the assumption that grasping dominates the traveling costs, simply iterate through all optimal FVS sets and then compute the subsequent minimum distance. After all such solutions are obtained, the optimal among these are chosen.

In practice, the resulting improvement in solution quality by iterating through all optimal FVS sets is negligible, as the returned solutions are often very close to optimal.

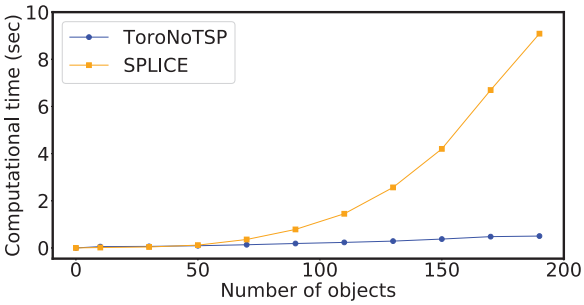


Fig. 10. Computational time of TORONOTSP and *SPLICE*.

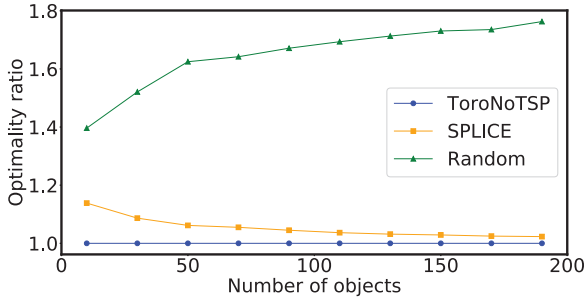


Fig. 11. Optimality ratio of TORONOTSP, *SPLICE*, and a random selection method.

This has been evaluated in small problem sizes, where the enumeration of optimal FVS sets is feasible.

6. Performance evaluation of simulations

All simulations are executed on an Intel® Core™ i7-6900K CPU with 32 GB RAM at 2133 MHz. Concorde (Applegate et al., 2007) is used for solving the TSP and Gurobi 6.5.1 (Gurobi Optimization, 2016) for ILP models.

6.1. TORO-NO: minimizing the travel distance

To evaluate the effectiveness of TORONOTSP, random TORO-NO instances are generated in which the number of objects varies. For each choice of number of objects, 100 instances are tried and the average is taken. The evaluation is mainly based on two metrics: average computational time (over 100 instances) and optimality ratio, which is defined as follows. Denoting c^i_{opt} and c^i_{alg} as the cost of the optimal solution and the current solution of the ith problem instance, respectively, the optimality ratio of the current algorithm is calculated as $(\sum_{i=1}^{100} c^i_{alg}/c^i_{opt})/100$. Although TORONOTSP works on thousands of objects (it takes less than 30 seconds for TORONOTSP to solve instances with 2,500 objects), the evaluation is limited to 200 objects[1].

TORONOTSP is compared with *SPLICE* (Treleaven et al., 2013), which is introduced in Section 4. As shown in Figure 10, it takes less than a second for TORONOTSP to compute the distance optimal pick-and-place action set. Figure 11 illustrates the solution quality of TORONOTSP, *SPLICE*, and an algorithm that picks a random feasible solution. Note that the random feasible solution generally has poor quality. As an asymptotically optimal algorithm, *SPLICE* does well as the number of objects increases, but under-performs compared with TORONOTSP. In conclusion, TORONOTSP provides the best performance on both computational time and optimality for practical sized TORO-NO problems.

For the unlabeled case (TORO-UNO), the same experiments are carried out. The results appear in Table 1. Note that *SPLICE* no longer applies. The last line of the table is the optimality ratio of random solutions, included for reference purposes. For larger cases, the TSP-based method is able to solve for over 500 objects in 30 seconds.

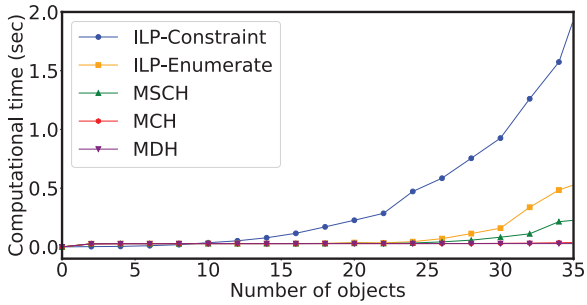


Fig. 12. Computational time of various FVS algorithms.

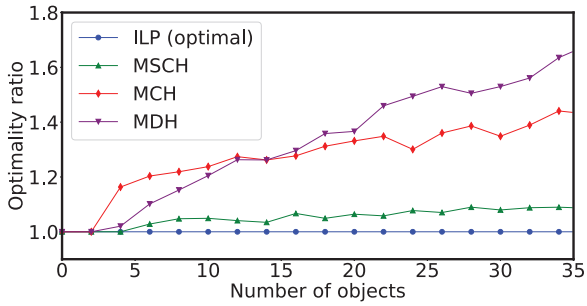


Fig. 13. Optimality ratio of various FVS algorithms compared with the optimal ILP-based methods.

6.2. TORO: minimizing the number of grasps

To evaluate different FVS minimization methods, dependency graphs are generated by capping the average degree and maximum degree for a fixed object count. To evaluate the computational time, the average degree is set to two and the maximum degree is set to four, which creates significant dependencies. The computational time comparison is given in Figure 12 (averaged over 100 runs per data point). Although exact ILP-based methods took more time than heuristics, they can solve optimally for over 30 objects in just a few seconds, which makes them very practical.

When it comes to performance (Figure 13), ILP-based methods have no competition. Interestingly, the simple cycle-based method (MSCH) also works quite well and may be useful in place of ILP-based methods for larger problems, given that MSCH runs faster.

The performance is also affected by the average degree for each node, which is directly linked to the complexity

Table 1. Evaluation of the TSP model for the unlabeled case.

Number of objects	10	50	100	200
Computational time (s)	0.04	0.58	2.43	7.30
Optimality of random solution	1.94	3.72	4.92	6.01

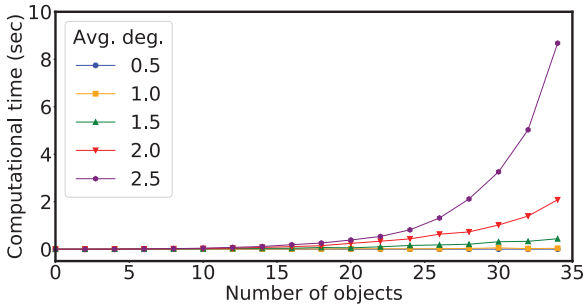


Fig. 14. Computational time of ILP-Constraint under varying G_{dep} average degree. Maximum degree is capped at twice the average degree.

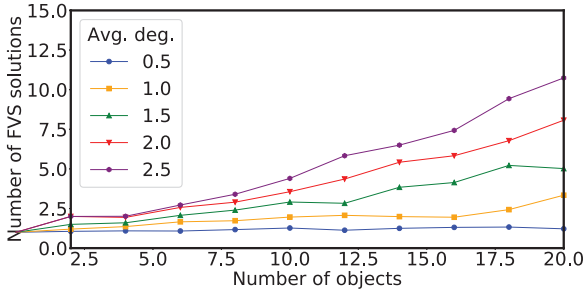


Fig. 15. The number of optimal FVS solutions in expectation.

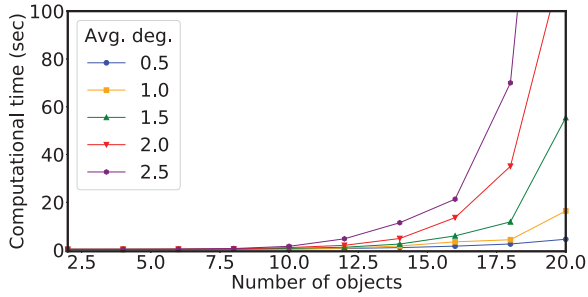


Fig. 16. The total computational time for TOROFVSSINGLE().

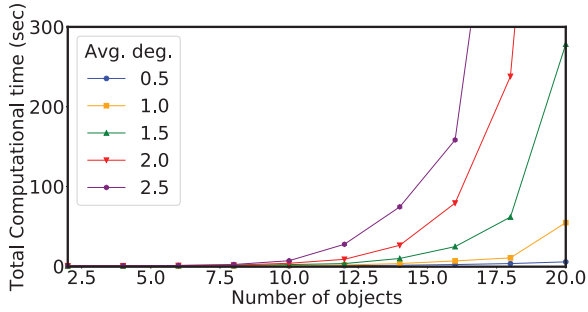


Fig. 17. The computational time to produce a global optimal solution for TORO.

of G_{dep}. Fixating on the ILP-Constraint algorithm, experiments with an average degree of 0.5–2.5 are included (2.5 average degree yields rather constrained dependency graphs). As can be observed from Figure 14, for up to 35 objects, an optimal FVS can be readily computed in a few seconds.

Finally, this section emphasizes an observation regarding the number of optimal FVS sets (Figure 15). By disabling FVSs that are already obtained in subsequent runs, all FVSs for a given problem can be exhaustively enumerated. The number of optimal FVSs turns out to be fairly limited.

6.3. TORO: overall performance

The computational time for the entire TOROFVSSINGLE() is provided in Figure 16. Observe that FVS computation takes almost no time in comparison with the distance minimization step. As expected, higher average degrees in G_{dep} make the computation harder.

Running TOROFVSSINGLE() together with FVS enumeration, a global optimal solution is computed for TORO under the assumption that the grasp/release costs dominate. Only solutions with an optimal FVS are considered. The computation time is provided in Figure 17. The result shows that it becomes costly to compute the global optimal solution as the number of objects go beyond 15 for dense setups. It is empirically observed that for the same problem instance and different optimal FVSs, the minimum distance computed by MINDIST() in Algorithm 2 has less than 5% variance. This suggests that running TOROFVSSINGLE() just once should yield a solution that is very close to being the global optimum.

7. Physical experiments

This section demonstrates an implementation of the algorithms introduced in this paper on a hardware platform (Figure 18). The physical experiments were conducted on three different tabletop scenarios. For each scenario, multiple problem instances were generated to compare the efficiency of the solutions produced by the proposed algorithms relative to alternatives, such as random and greedy solutions.

To apply object rearrangement algorithms in the physical world, it is first necessary to identify the problem parameters corresponding to the formulation of the TORO problem. Specifically, the algorithms are expecting as input a start

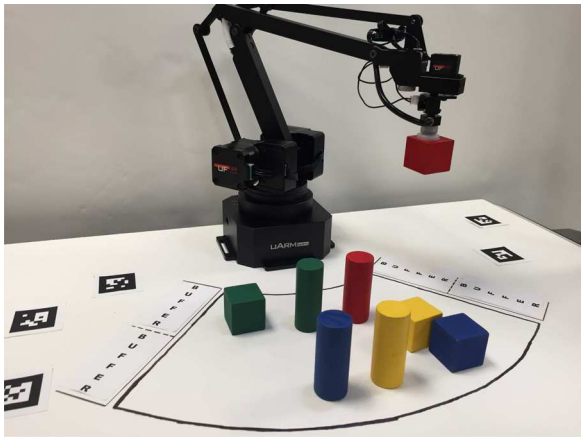


Fig. 18. A snapshot of the physical system carrying out a solution generated by the algorithms presented in this paper for a tabletop rearrangement scenario.

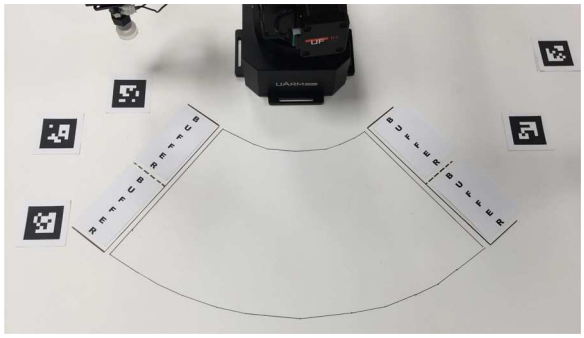
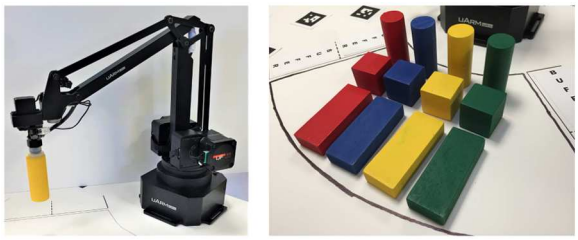


Fig. 19. A tabletop environment where the uArm Swift Pro's workspace is the area between $[45°, 135°]$ at a distance ranging between 140mm and 280mm from the base.



(a) (b)

Fig. 20. (a) The UFACTORY uArm Swift Pro grasping one of the labeled (colored) cylindrical objects. (b) An arrangement of 12 objects of various colors and geometries.

and a goal arrangement on a tabletop. The goal arrangement is predefined, whereas the objects are in an initial, arbitrary state. The physical platform first detects the start arrangement of the objects before executing a solution for rearranging the objects into the desired goal arrangement.

7.1. Hardware setup

The experimental setup is composed of five components.

1. Tabletop. The tabletop is a planar surface where the desktop manipulator and objects rest. Clearly defined contours are drawn denoting the projections of the manipulator's reachable area (i.e. workspace), including predefined buffers within this reachable area, as shown in Figure 19.

2. Manipulator. The manipulator is a UFACTORY uArm Swift Pro[2] (Figure 20(a)), which is an inexpensive but versatile desktop manipulator capable of performing repeatable actions with a precision of 0.2 mm. The uArm Swift Pro's versatility is largely due to the variety of end-effectors that can be equipped. The suction cup end-effector is utilized to achieve overhand grasps in the forthcoming challenges. Although the uArm Swift Pro is not built for industrial applications, it has enough precision to perform the experimental tasks described in this section.

3. Camera: The Logitech©webcam C920[3] is a consumer-grade webcam capable of Full HD video recording (1080p, 1,920 × 1,080 pixels) that is used for object pose detection in the experiments. The camera is mounted above the tabletop, such that its field of view (Figure 19) captures the entire workspace. The pre-calibration eliminates lens distortion and also renders brighter, saturated images, which makes pose detection easier.

4. Marker detection "Chilitags" is a cross-platform software library for the detection and identification of two-dimensional fiducial markers (Bonnard et al., 2013). Physical markers placed in the view of the imaging system are used as a point of reference/measure for surrounding objects. In the case of Chilitags, that object is a physical marker added to the environment. As seen in Figure 19, the experiments employ five tags, each placed at a known pose on the tabletop. This knowledge facilitates the computation of the 2D transformation between the manipulator's frame of reference and the camera's frame of reference.

5. Objects There are several objects that the manipulator interacts with during the experiments (Figure 20(b)). These objects are identifiable according to their shape and color

$$\{\text{cube, cylinder, orthotope}\} \times \{\text{red, blue, green, yellow}\}$$

resulting in 12 uniquely identifiable objects. The center of each object on the tabletop and its orientation define its configuration.

7.2. Object pose detection

The predetermined goal configuration of the objects is defined relative to the manipulator's frame of reference. The initial configuration is unknown and is determined online via the overhead C920 camera. Once the objects are detected, their observed configuration undergo a transformation to the manipulator's frame of reference.

The pose estimation method appears in Algorithm 3. In line 2, an image is taken by the camera. Line 3 utilizes *Gaussian blur* (also know as Gaussian smoothing) to reduce

Algorithm 3: OBJECTPOSEESTIMATION

```
1  objects ← {}
2  img ← CAMERACAPTURE( )
3  img ← GAUSSIANBLUR( img)
4  for color ∈ {red, blue, yellow, green} do
5      contours ← FINDCONTOURS( img, color)
6      for contour ∈ contours do
7          shape ← DETECTSHAPE( contour)
8          if shape ∈ {cube, cylinder, orthotope} then
9              position ←
                 CENTER( MINENCLOSINGCIRCLE( contour) )
10             orientation ←
                 PCA( MINAREARECT( contour) )
11             pose ← UPDATE( position, orientation, shape)
12             if INWORKSPACE(pose) then
13                 objects ← objects ∪ {( shape, color, pose) }
14 return objects
```

image noise. For each of the predefined colors (i.e. red, blue, yellow, green), the area(s) of the image matching the current color are extracted and further smoothed by *morphological transformations*, including *erosion* and *dilation*. The contours of these areas, which describe the top sides of objects, are then calculated (line 5). Each contour is examined to determine whether or not it corresponds to one of the objects in the scene (line 7). For each of the contours corresponding to an object in the scene, several operations need to occur to determine the pose of the object. The 2D point component of the pose as it appears in the camera is then determined by computing the center of the contour's minimum enclosing circle (line 9), whereas the orientation of the object in the camera's frame is extracted via *principle component analysis* over the minimum area rectangle containing the contour (line 10). Line 11 updates the 2D point component of the pose in the camera's frame to accurately reflect position of the object relative to the manipulator, taking into account the current shape geometry. The 2D perspective transformation between camera frame and robot frame is pre-computed using the marker detection software. The poses of the tags in the robot's frame are fixed, but the poses of the tags in the camera's frame are automatically detected at runtime.

7.3. Experimental validation

This section presents three tabletop object rearrangement scenarios that can be performed via overhand pick-and-place actions. For each scenario, specific problem instances have been provided that are solvable. The specific algorithm is determined at run-time, and corresponds to whether there is overlap between the start and goal object configurations. If a subset of the start and goal configurations overlap, the problem may require the use of external buffer(s). The number of extra pick-and-place actions and the corresponding number of buffers necessary to carry out the task can be determined via the dependency graph.

All of the generated solutions by the proposed methods perform an optimal number of pick-and-place actions. A solution is optimal with respect to the travel distance of the end-effector when the external buffers are not utilized. Problem instances that require the use of an external buffer(s) remain near-optimal with respect to the travel distance of the end-effector.

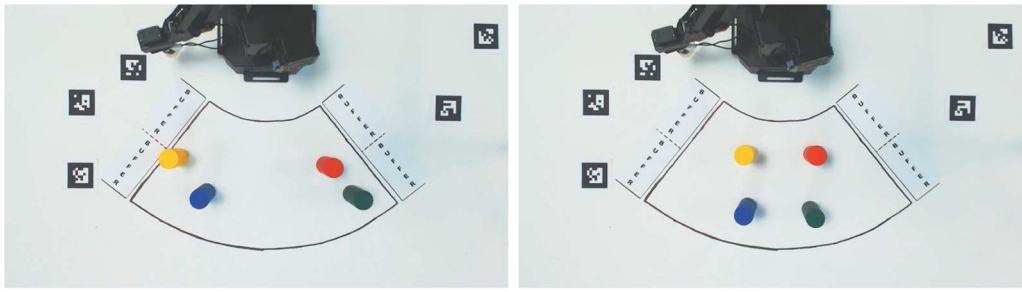
For each problem instance, a feasible solution is also generated by either a random algorithm (for `TORO-NO`) or a greedy algorithm (for `TORO` [4]). The execution time for the different solutions are then measured and compared.

Scenario 1: Cylinders without overlap This task requires the manipulator arm to transport four uniquely identifiable cylinders from a random start configuration to a fixed goal configuration. Figure 21(a) and (b) illustrate one such problem instance where the start and goal configurations do not overlap, indicating that this is a `TORO-NO` instance. As `TORO-NO` does not necessitate the use of any external buffers, the manipulator performs only four pick-and-place operations, corresponding to the number of cylinders in the scene. Appendix E provides the *distance optimal sequence* pick-and-place operations of this instance.

To illustrate the efficiency of solutions generated by TORONOTSP, 10 different problem instances are generated. They are solved by both TORONOTSP and random grasping sequences. Note that any permutation of objects is a feasible solution. The result is presented in Table 2. The first column in this table specifies different problem instances. The *raw* columns measure the total execution time, which contains grasps, releases, and transportation. As the time for grasps and releases is the same for all the feasible solutions for a `TORO-NO` instance, the amount of time (45.23 s) to perform four in-place pick-and-place actions is then deducted from total execution time. These results appear in the *tran* columns. Moreover, the difference between the execution time of the algorithms appear in the *difference* column.

From the empirical data provided in Table 2, random solution strategies for this problem setup incur on average a 9.6% increase in transportation time compared with the solution generated by TORONOTSP.

Remark 6. The difference of execution time between a random solution and the optimal solution is generally proportional to, but not linearly dependent on the transition cost defined in this paper, which is based on the Euclidean distance between poses in two dimensions. This is due to the manipulator model. In practice, the path that the end-effector travels does not necessarily need to be along piecewise linear shortest paths (straight lines). In addition, the second-order term (acceleration) varies, contributing to a non-static velocity, which is not captured by the tabletop model described herein.



(a) (b)

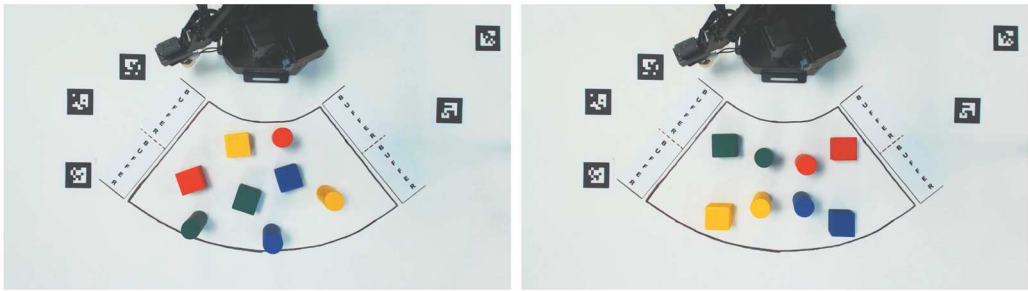
Fig. 21. Problem instance of Scenario 1: (a) start arrangement; (b) goal arrangement.

Table 2. Experimental results showing a comparison of the execution time between TORONOTSP and a random feasible solution for ten problem instances. The average computational time for TOROFVSSINGLE is 0.03 seconds.

Instance	Execution time (s)				
	TORONOTSP		Random solution		Difference
	Raw	Tran	Raw	Tran	
1	65.04	19.81	68.50	23.27	3.46
2	70.44	25.21	75.38	30.15	4.94
3	67.11	21.88	69.85	24.62	2.74
4	69.34	24.11	73.09	27.86	3.75
5	70.67	25.44	73.04	27.81	2.37
6	71.13	25.90	73.37	28.14	2.24
7	73.40	28.17	73.66	28.43	0.26
8	67.95	22.72	67.97	22.74	0.02
9	66.62	21.39	68.21	22.98	1.59
10	66.88	21.65	68.22	22.99	1.34
Average	68.86	23.63	71.13	25.90	2.27

Table 3. Experimental results showing a comparison of the execution time between TOROFVSSINGLE and a greedy solution for two scenarios.

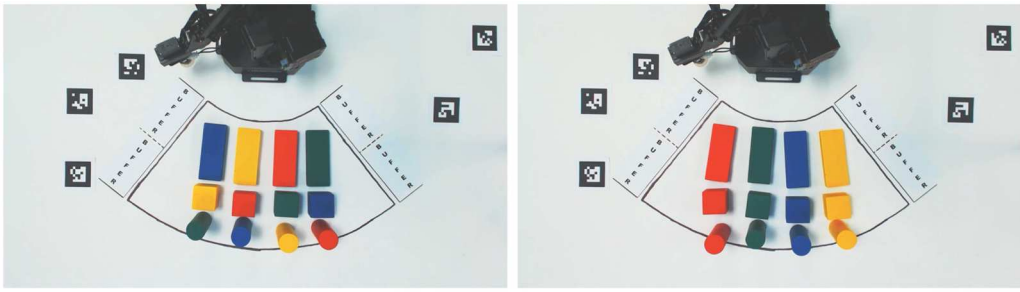
Scenario	Execution time (sec)		Computational time (s)		Number of actions	
	TOROFVSSINGLE	Greedy algorithm	TOROFVSSINGLE	Greedy algorithm	TOROFVSSINGLE	Greedy algorithm
2	165.57	190.02	0.29	< 0.001	9	10
3	299.95	405.16	10.71	< 0.001	16	20



(a) (b)

Fig. 22. Problem instance of Scenario 2: (a) start arrangement; (b) goal arrangement.

Scenario 2: Cylinders and cubes In this scenario, the arm is tasked with rearranging eight objects (four cylinders and four cubes) that are initially scattered throughout the workspace, while the desired goal configuration is predetermined. Figure 22(a) and (b) show one such instance. Owing to overlap between the start and goal configurations, this is a TORO instance and thus uses the TOROFVSSINGLE algorithm, making use of the external buffers available to the manipulator. In the solution of TOROFVSSINGLE on this particular instance, the manipulator uses one of the



(a) (b)

Fig. 23. Problem instance for Scenario 3 containing 12 objects: (a) start arrangement; (b) goal arrangement.

available buffer locations to perform the rearrangement. The movement of an object to an external buffer results in a total of nine pick-and-place actions.

TOROFVSSINGLE is compared with a greedy method, which solves the problems sequentially by first removing the dependencies for one object and then moving it to its goal. As shown in Table 3, the greedy algorithm returns 1 extra grasp and takes 24.45 s of additional execution time, compared with TOROFVSSINGLE.

Scenario 3: Cylinders, cubes, orthotopes This scenario involves rearranging 12 objects (four of each type of cylinders, cubes, and orthotopes) within the manipulator's workspace. Objects of identical geometry are aligned in front of the manipulator, ordered by increasing height (i.e. orthotopes, cubes, cylinders). Thus, start and goal configurations differ by permutations of color amongst objects of the same shape. The instance shown in Figure 23(a) and (b) utilizes three of the available buffer locations as it performs 16 pick-and-place actions necessary to solve the task.

As the number of objects increases, the difference between the solution quality of TOROFVSSINGLE and the greedy algorithm becomes larger. The solution of greedy algorithm has 4 extra grasps and 105.21 s more execution time compared with the sub-optimal solution generated by TOROFVSSINGLE.

8. Conclusion

This paper had studied the combinatorial structure inherent in tabletop object rearrangement problems. For TORO-NO and TORO-UNO, it has been shown that Euclidean-TSP can be reduced to them, establishing their NP-hardness. More importantly, TORO-NO and TORO-UNO can be reduced to TSP with little overhead, thus establishing that they have similar computational complexity and lead to an efficient solution scheme. Similarly, an equivalence has been established between dependence breaking of TORO and FVS, which is APX-hard. The equivalence enables subsequent ILP-based methods for effectively and optimally solving TORO instances containing tens of objects with overlapping starts and goals.

The methods and algorithms in this paper serve as an initial foundation for solving complex rearrangement tasks on tabletops. Many interesting problems remain open in this area; two are highlighted here. The current paper assumes the availability of *external* buffers, which are separated from the workspace occupied by the objects, demanding additional movement from the end-effector. In practice, it can be beneficial to dynamically locate buffers that are close by, which may be tackled through effective sampling methods. Furthermore, the scenarios addressed only static settings whereas many industrial settings require solving a more dynamics problem in which the objects to be rearranged do not remain still with respect to the robot base.

Acknowledgment

The authors would like to thank uFactory for supplying the uArm Swift Pro robot that was used in the hardware-based evaluation. The authors would also like to recognize the contribution of Jiakun Lyu, Zhejiang University, China, for his help with the object pose detection used in the hardware experiments.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the NSF (award numbers IIS-1617744, IIS-1451737, CCF-1330789, and IIS-1734419) as well as internal support by Rutgers University. Any opinions or findings expressed in this paper do not necessarily reflect the views of the sponsors.

ORCID iDs

Shuai D Han https://orcid.org/0000-0001-7741-2378
Nicholas M Stiffler https://orcid.org/0000-0002-0164-1809

Notes

1. State-of-the-art Delta robots have comparable abilities. For example, the Kawasaki YF03 Delta Robot is capable of performing 222 pick-and-place actions per minute (1 kg objects).
2. See http://www.ufactory.cc/
3. See https://www.logitech.com/en-us/product/hd-pro-webcam-c920

4. The solution produced by the TOROFVSSINGLE algorithm uses the solution returned by the Gurobi ILP solver after 10 seconds of compute time. Empirically, any further computation only minimizes the transition time between poses at the expense of increased computation. Note that this does not affect the number of grasps that remain optimal.

References

Anily S and Hassin R (1992) The swapping problem. *Networks* 22(4): 419–433.

Applegate DL, Bixby RE, Chvatal V and Cook WJ (2007) *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ: Princeton University Press.

Aronov B, de Berg M, van den Stappen AF, Švestka P and Vleugels J (1999) Motion planning for multiple robots. *Discrete and Computational Geometry* 22(4): 505–525.

Auletta V, Monti A, Parente D and Persiano G (1999) A linear time algorithm for the feasibility of pebble motion on trees. *Algorthmica* 23: 223–245.

Bafna V, Berman P and Fujito T (1999) A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics* 12(3): 289–297.

Baharev A, Schichl H and Neumaier A (2015) An exact method for the minimum feedback arc set problem. University of Vienna. Available at: http://www.mat.univie.ac.at/ neum/ms/minimum_feedback_arc_set.pdf

Ben-Shahar O and Rivlin E (1998) Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation* 14(4): 549–565.

Berbeglia G, Cordeau JF, Gribkovskaia I and Laporte G (2007) Static pickup and delivery problems: a classification scheme and survey. *Top* 15(1): 1–31.

Berbeglia G, Cordeau JF and Laporte G (2010) Dynamic pickup and delivery problems. *European Journal of Operational Research* 202(1): 8–15.

Berenson D, Srinivasa SS and Kuffner JJ (2012) Task space regions: a framework for pose-constrained manipulation planning. *The International Journal of Robotics Research* 30(12): 1435–1460.

Beullens P, van Oudheusden D and van Wassenhove LN (2004) Collection and vehicle routing issues in reverse logistics. In: *Reverse Logistics*. New York: Springer, pp. 95–134.

Bohg J, Morales A, Asfour T and Kragic D (2014) Data-driven grasp synthesis - a survey. In: *IEEE Transactions on Robotics* 30: 289–309.

Bonnard Q, Lemaignan S, Zufferey G, et al. (2013) Chilitags 2: robust fiducial markers for augmented reality and robotics. Available at: http://chili.epfl.ch/software.

Calinescu G, Dumitrescu A and Pach J (2008) Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics* 22(1): 124–138.

Cambon S, Alami R and Gravot F (2009) A hybrid approach to intricate motion, manipulation, and task planning. *The International Journal of Robotics Research* 28: 104–126.

Chen PC and Hwang YK (1991) Practical path planning among movable obstacles. In: *Proceedings IEEE International Conference on Robotics and Automation*, pp. 444–449.

Christofides N and Eilon S (1969) An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society* 20(3): 309–318.

Ciocarlie M and Allen P (2009) Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research* 28: 851–867.

Cohen JB, Chitta S and Likhachev M (2013) Single- and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research* 33: 305–320.

Cosgun A, Hermans T, Emeli V and Stilman M (2011) Push planning for object placement on cluttered table surfaces. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Dantam NT, Kingston ZK, Chaudhuri S and Kavraki LE (2016) Incremental task and motion planning: a constraint-based approach. In: *Proceedings Robotics: Science and Systems*.

Demaine E, O'Rourke J and Demaine ML (2000) Pushpush and push-1 are NP-hard in 2D. In: *Proceedings Candadian Conference on Computational Geometry*, pp. 211–219.

Dinur I and Safra S (2005) On the hardness of approximating minimum vertex cover. *Annals of Mathematics* 162: 439–485.

Dogar MR and Srinivasa SS (2011) A framework for push-grasping in clutter. In: *Proceedings Robotics: Science and Systems*.

Even G, Naor JS, Schieber B and Sudan M (1998) Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica* 20(2): 151–174.

Frederickson GN and Guan DJ (1993) Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms* 15(1): 29–60.

Frederickson GN, Hecht MS and Kim CE (1976) Approximation algorithms for some routing problems. In: *17th Annual Symposium on Foundations of Computer Science (SFCS 1976)*. IEEE, pp. 216–227.

Garey MR and Johnson DS (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2014) FFRob: an efficient heuristic for task and motion planning. In: *Proceedings Workshop on the Algorithmic Foundations of Robotics*.

Gharbi M, Lallement R and Alami R (2015) Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6360–6365.

Goraly G and Hassin R (2010) Multi-color pebble motion on graphs. *Algorthmica* 58(3): 610–636.

Gribkovskaia I, Halskau Ø, Laporte G and Vlček M (2007) General solutions to the single vehicle routing problem with pick-ups and deliveries. *European Journal of Operational Research* 180(2): 568–584.

Gurobi Optimization I (2016) Gurobi Optimizer Reference Manual. Available at: http://www.gurobi.com.

Halperin D, Latombe JC and Wilson RH (2000) A general framework for assembly planning: the motion space approach. *Algorthmica* 26(3–4): 577–601.

Han SD, Stiffler NM, Krontiris A, Bekris KE and Yu J (2017) High-quality tabletop rearrangement with overhand grasps: hardness results and fast methods. In: *Proceeedings Robotics: Science and Systems*, Boston, MA.

Hauser K (2014) The minimum constraint removal problem with three robotics applications. *The International Journal of Robotics Research* 33(1): 5–17.

Havur G, Ozbilgin G, Erdem E and Patoglu V (2014) Geometric rearrangement of multiple moveable objects on cluttered surfaces: a hybrid reasoning approach. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Hoff A and Løkketangen A (2006) Creating lasso-solutions for the traveling salesman problem with pickup and delivery by tabu search. *Central European Journal of Operations Research* 14(2): 125–140.

Karp RM (1972) Reducibility among combinatorial problems. In: *Complexity of Computer Computations*. New York: Springer, pp. 85–103.

Kornhauser D, Miller G and Spirakis P (1984) Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: *Proceedings IEEE Symposium on Foundations of Computer Science*, pp. 241–250.

Krontiris A and Bekris KE (2015) Dealing with difficult instances of object rearrangement. In: *Proceedings Robotics: Science and Systems*, Rome, Italy.

Krontiris A and Bekris KE (2016) Efficiently solving general rearrangement tasks: a fast extension primitive for an incremental sampling-based planner. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Krontiris A, Shome R, Dobson A, Kimmel A and Bekris KE (2014) Rearranging similar objects with a manipulator using pebble graphs. In: *Proceedings IEEE International Conference on Humanoid Robotics*, Madrid, Spain.

Laporte G (1992) The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59(3): 345–358.

Leroy S, Laumond JP and Siméon T (1999) Multiple path coordination for mobile robots: a geometric algorithm. In: *Proceedings International Joint Conferences on Artificial Intelligence*, pp. 1118–1123.

Monien B and Speckenmeyer E (1985) Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica* 22(1): 115–123.

Nieuwenhuisen D, van der Stappen AF and Overmars MH (2006) An Effective Framework for Path Planning amidst Movable Obstacles. In: *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

Ota J (2004) Rearrangement planning of multiple movable objects. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Papadimitriou CH (1977) The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4(3): 237–244.

Plaku E and Hager G (2010) Sampling-based motion and symbolic action planning with geometric and differential constraints. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Rohmer E, Singh SPN and Freese M (2013) V-REP: a versatile and scalable robot simulation framework. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Sharon G, Stern R, Felner A and Sturtevant NR (2015) Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219: 40–66.

Siméon T, Laumond JP, Cortés J and Sahbani A (2004) Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research* 23: 729–746.

Solovey K and Halperin D (2015) On the hardness of unlabeled multi-robot motion planning. In: *Proceedings Robotics: Science and Systems*.

Solovey K, Yu J, Zamir O and Halperin D (2015) Motion planning for unlabeled discs with optimality guarantees. In: *Proceedings Robotics: Science and Systems*.

Srivastava S, Fang E, Riano L, Chitnis R, Russell S and Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Stilman M, Schamburek J, Kuffner JJ and Asfour T (2007) Manipulation planning among movable obstacles. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Sundaram S, Remmler I and Amato NM (2001) Disassembly sequencing using a motion planning approach. In: *Proceedings IEEE International Conference on Robotics and Automation*, Washington, DC, pp. 1475–1480.

Treleaven K, Pavone M and Frazzoli E (2013) Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems. *IEEE Transactions on Automatic Control* 58(9): 2261–2276.

van den Berg J and Overmars M (2005) Prioritized motion planning for multiple robots. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2217–2222.

van den Berg J, Snoeyink J, Lin M and Manocha D (2009) Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: *Proceedings Robotics: Science and Systems*.

van den Berg J, Stilman M, Kuffner JJ, Lin M and Manocha D (2008) Path planning among movable obstacles: a probabilistically complete approach. In: *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

Vega-Brown W and Roy N (2016) Asymptotically optimal planning under piecewise-analytic constraints. In: *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

Wagner G, Kang M and Choset H (2012) Probabilistic path planning for multiple robots with subdimensional expansion. In: *Proceedings IEEE International Conference on Robotics and Automation*.

Wilfong G (1991) Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence* 3: 131–150.

Wilson RH and Latombe JC (1994) Geometric reasoning about mechanical assembly. *Artificial Intelligence* 71(2): 371–396.

Yu J and LaValle SM (2012) Multi-agent path planning and network flow. In: *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

Yu J and LaValle SM (2016) Optimal multi-robot path planning on graphs: complete algorithms and effective heuristics. *IEEE Transactions on Robotics* 32(5): 1163–1177.

Zucker M, Ratliff N, Dragan A, et al. (2013) CHOMP: covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research* 32: 1164–1193.

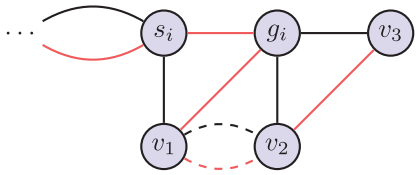


Fig. 24. Augmenting a path in a TORO-UNO solution. The black segments indicate a previous path, whereas the red segments demonstrate the augmented path.

Appendix A. Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of multimedia extension

Extension	Media type	Description
1	Video	Complete executions for each of the three scenarios described in Section 7.3

Appendix B. Proof for cost-optimal TORO-UNO

Proof of Theorem IV.2. Again, reduce from Euclidean-TSP. The same TSP instance from the proof of Theorem IV.1 is used. The conversion to TORO-NO and the process to obtain a TORO-UNO instance are also similar, with the exception being that edges s_ig_i are not required to be used in a solution; this makes the labeled case become unlabeled.

The argument is that the cost-optimal solution of the TORO-UNO instance also yields an optimal solution to the original Euclidean-TSP tour. This is accomplished by showing that an optimal solution to the TORO-NO instance has essentially the same cost as the TORO-UNO instance. To see that this is the case, assume that an optimal solution (tour) path to the reduced TORO-UNO problem is given. Let the path have a total length (cost) of $D_{opt}^{\text{TORO-UNO}}$. Let s_ig_i be the first such edge that is not in the TORO-UNO solution. Because the path is a tour, following s_i along the path will eventually reach g_i. The resulting path will have the form $s_iv_1 \ldots v_2g_iv_3$, i.e. the black path in Figure 24.

Upon the observation of such a partial solution, proceed to make the augmentation and replace the path with the new one (red path in Figure 24). Because $s_ig_i \ll 1/(4n)$, the potential increase in path length is bounded by (note that v_2v_3 is shorter than the additive length of v_2g_i and g_iv_3)

$$\|s_ig_i\|_2 + \|g_iv_1\|_2 - \|s_iv_1\|_2 \leq 2\varepsilon \ll 1/(2n)$$

After at most n such augmentations, an optimal TORO-UNO solution is converted into a TORO-NO solution. The TORO-NO solution has a cost increase of at most $n * 1/(2n) = 1/2$. The TORO-NO solution can then be converted into a solution of the Euclidean-TSP problem, which will not increase the cost. Thus, a TORO-UNO solution can be converted to a corresponding Euclidean-TSP solution with a cost addition of less than $1/2$. Let the Euclidean-TSP solution obtained in this manner have a total cost of D', then

$$D' < D_{opt}^{\text{TORO-UNO}} + \frac{1}{2} \tag{4}$$

Now again let the optimal Euclidean-TSP solution have a cost of D_{opt}. The solution can be converted to a TORO-NO solution with a total cost of less than $D_{opt} + 1/4$. The TORO-NO solution is also a solution to the TORO-UNO problem. That is, for this new TORO-UNO solution, the cost is

$$D^{\text{TORO-UNO}} < D_{opt} + \frac{1}{4} \tag{5}$$

Now, if $D' > D_{opt}$, then $D' \geq D_{opt} + 1$. Putting this together with (4) and (5),

$$D^{\text{TORO-UNO}} < D_{opt} + \frac{1}{4} \leq D' - \frac{3}{4} \leq D_{opt}^{\text{TORO-UNO}} - \frac{1}{4}$$

which is a contradiction. Therefore, $D' > D_{opt}$ cannot be true. Therefore, a cost-optimal TORO-UNO solution yields an optimal solution to the original Euclidean-TSP problem. This shows that TORO-UNO is at least as hard as Euclidean-TSP. □

Appendix C. Exact ILP-based algorithms for finding optimal FVS

To compute the exact solution, the problem is modeled as an ILP problem, and then solved using LP solvers, e.g. Gurobi TSP Solver Gurobi Optimization (2016). In this paper, two different ILP models are used, which are similar to the models introduced by (Baharev et al., 2015, sections 3.1 and 3.2).

1. **ILP-Constraint.** By splitting all vertices $o_i \in G_{dep}$ to o_i^{in} and o_i^{out}, a new graph $G_{arc}(V_{arc}, E_{arc})$ is constructed, where $V_{arc} = \{o_1^{in}, o_1^{out}, \ldots, o_n^{in}, o_n^{out}\}$, and $(o_i^{out}, o_j^{in}) \in E_{arc}$ iff $(o_i, o_j) \in A_{dep}$. By adding extra edges (o_i^{in}, o_i^{out}) for all $1 \leq i \leq n$ to E_{arc}, problem is transformed into a *minimum feedback arc set* problem, where the objective is to find a minimum set of arcs to make G_{arc} acyclic. Moreover, every edge in this set ends at o_i^{in} or starts from o_i^{out} can be replaced by (o_i^{in}, o_i^{out}), which denotes a vertex o_i in G_{dep}, without changing the feasibility of the solution.
   The next step is to find a minimum cost ordering π^* of the nodes in G_{arc}. Let $c_{i,j} = 1$ if edge $(i,j) \in E_{arc}$, whereas $c_{i,j} = 0$ if edge $(i,j) \notin E_{arc}$. Furthermore, let binary variables $y_{i,j}$ associate the ordering of $i,j \in \pi$,

where $y_{i,j} = 0$ if i precedes j, or 1 if j precedes i. Suppose $|V_{arc}| = m$, the LP formulation is expressed as:

$$\min_{y} \quad \sum_{j=1}^{m}\Big(\sum_{k=1}^{j-1} c_{k,j}y_{k,j} + \sum_{l=j+1}^{n} c_{l,j}(1-y_{j,l})\Big)$$

$$\begin{aligned} \text{s.t.} \quad & y_{i,j}+y_{j,k}-y_{i,k} \le 1, \quad 1 \le i < j < k \le m \\ & -y_{i,j}-y_{j,k}+y_{i,k} \le 0, \quad 1 \le i < j < k \le m \end{aligned}$$

The solution arc set contains all the backward edges in π^*.

2. **ILP-Enumerate.** First find the set C of all the simple cycles in G_{dep}. A set of binary variables $V = \{v_1, \ldots, v_n\}$ is defined, each assigned to an object $o_i \in O$, the LP formulation is expressed as

$$\begin{aligned} \max_{v} \quad & \sum_{v_i \in V} v_i \\ \text{s.t.} \quad & \sum_{o_i \in C_j} v_i < |C_j|, \quad \forall C_j \in C \end{aligned}$$

Then the vertices in the minimum FVS are the objects whose corresponding variable v_i is zero in the solution of this LP model.

Appendix D. ILP model to find the shortest travel distance in TORO

Given a TORO instance with n objects $\mathcal{O} = \{o_1, \ldots, o_n\}$, without loss of generality, $B = \{o_1, \ldots, o_p\} \subset \mathcal{O}$ denotes the set of objects to be moved to buffers, which is calculated by the methods introduced in Section 5.2 and Appendix C. The maximum number of buffers to be used is denoted as $p = |B|$. The ILP model introduced in this section finds the distance-optimal solution amongst all candidates that moves objects in B to intermediate locations before moving them to goal configurations while employing the minimum number of grasps (i.e. $n + p$).

All variables in this ILP model are Boolean variables, and have two components, *nodes* and *edges*, where a node denotes the occupancy of a position on the tabletop at a specific time step, and an edge represents a movement of the manipulator between two positions. The variables appear as a repeated graph pattern in a discrete time domain $t \in \{0, \ldots, n+p\}$, where time step t correlates to the states of the system after the tth pick-and-place action.

Assume the following:

$$\begin{aligned} 1 \le i &\le n \\ 1 \le j &\le p \\ 1 \le k &\le p \\ 1 \le \ell &\le n \\ p < m &\le n \end{aligned}$$

D.1. Variables: Nodes

There are four kinds of nodes:

- $\{s_1^t, \ldots, s_n^t\}$, occupancy of start configurations; $s_i^t = 1$ indicates o_i is at its start configuration at time step t;
- $\{g_1^t, \ldots, g_n^t\}$, occupancy of goal configurations; $g_i^t = 1$ indicates o_i is at its goal configuration at time step t;
- $\{b_{11}^t, b_{12}^t, \ldots, b_{pp}^t\}$, occupancy of buffers; $b_{jk}^t = 1$ indicates o_j is in buffer b_k at time step t.
- $\{s_M, g_M\}$, occupancy of the manipulator's rest positions.

The value of the nodes when $t = 0$ and $t = n+p$ indicate the start and goal arrangement, respectively. Specifically,

$$\begin{aligned} s_i^0 = 1, \quad g_i^0 = 0, \quad b_{jk}^0 = 0 \\ s_i^{n+p} = 0, \quad g_i^{n+p} = 1, \quad b_{jk}^{n+p} = 0 \end{aligned}$$

D.2. Variables: Edges

Edges model the movement of the manipulator and are included in the cost function. An edge connects two nodes, e.g. node 1 and 2, and is denoted as $e(12)$. Edges in the ILP model are listed as follows.

- $e(s_m^t g_m^t)$, for $1 \le t \le n+p$. A positive value indicates a pick-and-place action that brings o_m from its start configuration to its goal configuration at time step t.
- $e(s_j^t b_{jk}^t)$, $e(b_{jk}^t g_j^t)$, for $1 \le t \le n+p$. A positive value indicates a pick-and-place action that brings o_j from its start configuration to buffer k, and from buffer k to its goal configuration.
- $e(g_i^t s_\ell^{t+1})$, $e(b_{jk}^t s_i^{t+1})$, $e(g_i^t b_{jk}^{t+1})$, for $1 \le t < n+p$, indicates a movement of the manipulator without an object being grasped.
- $e(s_M s_i^1)$, $e(g_i^{n+p} g_M)$, denote the target objects of the first and the last pick-and-place actions.

Note that because each edge is associated with a movement of the manipulator, the total travel distance of the end-effector can be modeled as the summation of the value (i.e. 0 or 1) of all edges multiplied with the cost associated with the edge. The objective of this ILP model is

$$\min \sum_{e} \text{cost}(e)$$

where $\text{cost}(e)$ denotes the distance between positions associated with nodes connected by edge e.

An illustration of the ILP model is provided in Figure 25.

D.3. Constraints

This appendix addresses the constraints that appear in the ILP model. These constraints update the value of variables, which ensures that the solution returned by the ILP model is able to be reduced to a feasible solution for the original TORO.

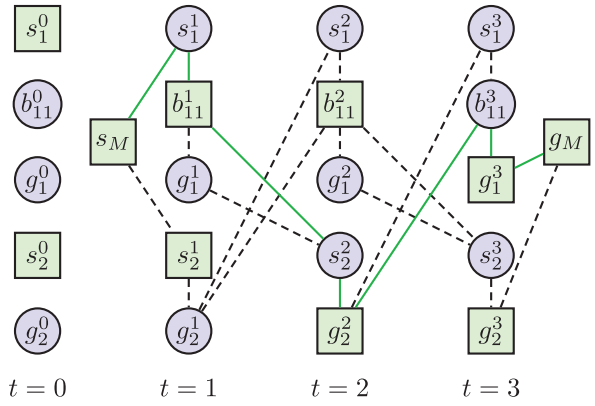


Fig. 25. An example of ILP model when $n=2, p=1$ as well as the value of variables after solving this model. The green square nodes and green solid edges indicate the value of variables is 1. The solution is interpreted as (i) bring o_1 to a buffer, (ii) bring o_2 to its goal, (iii) bring o_1 to its goal.

Begin by considering the constraints of the first pick-and-place action. To update the occupancy of start configurations:

$$\sum_{i=1}^{n} e(s_M s_i^1) = s_M = 1, s_i^1 = s_i^0 - e(s_M s_i^1)$$

To update the occupancy of buffers and goal configurations:

$$\sum_{k=1}^{p} e(s_j^1 b_{jk}^1) = e(s_M s_j^1), b_{jk}^1 = b_{jk}^0 + e(s_j^1 b_{jk}^1)$$
$$e(s_m^1 g_m^1) = e(s_M s_m^1), g_m^1 = g_m^0 + e(s_m^1 g_m^1)$$

After this step, the constraints for all other time steps can be modeled. Assume the following: $1 \le t \le n+p-1$.

The movement of the manipulator between time step t and $t+1$ correspond to

$$e(s_j^t b_{jk}^t) = \sum_{i=1}^{n} e(b_{jk}^t s_i^{t+1}) + \sum_{a=1}^{p}\sum_{b=1}^{p} e(b_{jk}^t b_{ab}^{t+1})$$
$$\sum_{j=1}^{p} e(b_{jk}^t g_j^t) = \sum_{i=1}^{n} e(g_j^t s_i^{t+1}) + \sum_{k=1}^{p}\sum_{l=1}^{p} e(g_j^t b_{kl}^{t+1})$$
$$e(s_m^t g_m^t) = \sum_{i=1}^{n} e(g_m^t s_i^{t+1}) + \sum_{k=1}^{p}\sum_{l=1}^{p} e(g_j^t b_{kl}^{t+1})$$

For time step $t+1$, constraints are imposed on the incoming edges from time step t, in order to avoid the scenario where the manipulator travels to a vacant location:

$$\sum_{i=1}^{n} e(g_i^t s_\ell^{t+1}) + \sum_{j=1}^{p}\sum_{k=1}^{p} e(b_{jk}^t s_\ell^{t+1}) \le s_\ell^t$$
$$\sum_{i=1}^{n} e(g_i^t b_{jk}^{t+1}) + \sum_{a=1}^{p}\sum_{b=1}^{p} e(b_{ab}^t b_{jk}^{t+1}) \le b_{jk}^t$$

Update the edges to simulate a pick-and-place action in time step $t+1$:

$$\sum_{k=1}^{p} e(s_j^{t+1} b_{jk}^{t+1}) = \sum_{i=1}^{n} e(g_i^t s_j^{t+1}) + \sum_{a=1}^{p}\sum_{b=1}^{p} e(b_{ab}^t s_j^{t+1})$$
$$e(b_{jk}^{t+1} g_j^{t+1}) = \sum_{i=1}^{n} e(g_i^t b_{jk}^{t+1}) + \sum_{a=1}^{p}\sum_{b=1}^{p} e(b_{ab}^t b_{jk}^{t+1})$$
$$e(s_m^{t+1} g_m^{t+1}) = \sum_{i=1}^{n} e(g_i^t s_m^{t+1}) + \sum_{j=1}^{p}\sum_{k=1}^{p} e(b_{jk}^t s_m^{t+1})$$

Update nodes in time step $t+1$:

$$s_i^{t+1} = s_i^t - \sum_{\ell=1}^{n} e(g_\ell^t s_i^{t+1}) - \sum_{j=1}^{p}\sum_{k=1}^{p} e(b_{jk}^t s_i^{t+1})$$
$$b_{jk}^{t+1} = b_{jk}^t + e(s_j^{t+1} b_{jk}^{t+1}) - \sum_{i=1}^{n} e(g_i^t b_{jk}^{t+1}) - \sum_{a=1}^{p}\sum_{b=1}^{p} e(b_{ab}^t b_{jk}^{t+1})$$
$$g_j^{t+1} = g_j^t + \sum_{k=1}^{p} e(b_{jk}^{t+1} g_j^{t+1})$$
$$g_m^{t+1} = g_m^t + e(s_m^{t+1} g_m^{t+1})$$

As each buffer is presented as multiple copies in each time step, one must make sure it is occupied by at most one object:

$$\sum_{j=1}^{p} b_{jk}^t \le 1$$

Update the dependencies in the dependency graph. Suppose s_i is in collision with g_ℓ, then

$$s_i^t + g_\ell^t \le 1$$

After employed all $n+p$ pick-and-place actions, the manipulator goes to the rest position g_M:

$$e(g_j^{n+p} g_M) = \sum_{k=1}^{p} e(b_{jk}^{n+p} g_j^{n+p})$$
$$e(g_m^{n+p} g_M) = e(s_m^{n+p} g_m^{n+p})$$
$$g_M = \sum_{j=1}^{p} e(g_j^{n+p} g_M) + \sum_{m=p+1}^{n} e(g_m^{n+p} g_M) = 1$$

Appendix E. Solution for a TORO-NO instance in Section 7.3

Figure 26 demonstrates the optimal solution for the problem instance shown in Figure 21. The sequence of pick-and-place actions over the colored cylinders are yellow, red, green, and blue.

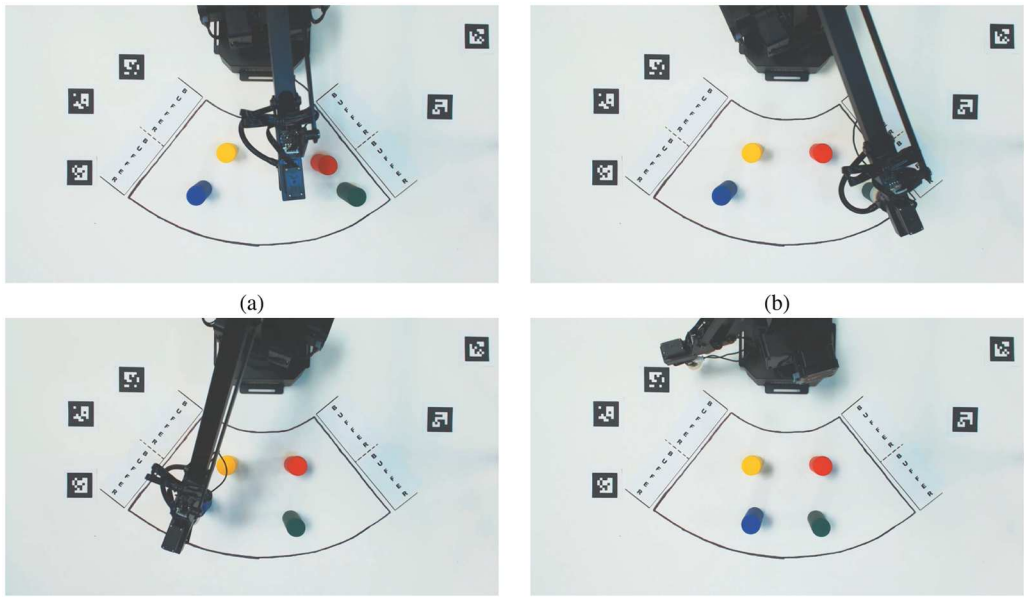


Fig. 26. Step-by-step solution for the TORO-NO instance in Figure 21.