Autonomous Robots (2019) 43:649–664
https://doi.org/10.1007/s10514-018-9748-z

Multi-robot grasp planning for sequential assembly operations

Mehmet Dogar[1] · Andrew Spielberg[2] · Stuart Baker[2] · Daniela Rus[2]

Received: 29 March 2017 / Accepted: 2 April 2018 / Published online: 16 April 2018

Abstract
This paper addresses the problem of finding robot configurations to grasp assembly parts during a sequence of collaborative assembly operations. We formulate the search for such configurations as a constraint satisfaction problem (CSP). *Collision constraints* in an operation and *transfer constraints* between operations determine the sets of feasible robot configurations. We show that solving the connected constraint graph with off-the-shelf CSP algorithms can quickly become infeasible even for a few sequential assembly operations. We present an algorithm which, through the assumption of feasible *regrasps*, divides the CSP into independent smaller problems that can be solved exponentially faster. The algorithm then uses local search techniques to improve this solution by removing a gradually increasing number of regrasps from the plan. The algorithm enables the user to stop the planner *anytime* and use the current best plan if the cost of removing regrasps from the plan exceeds the cost of executing those regrasps. We present simulation experiments to compare our algorithm's performance to a naive algorithm which directly solves the connected constraint graph. We also present a physical robot system which uses the output of our planner to grasp and bring parts together in assembly configurations.

1 Introduction

We are interested in multi-robot systems that can perform sequences of assembly operations to build complex structures. Each assembly operation in the sequence requires multiple robots to grasp multiple parts and bring them together in space in specific relative poses. We present an example in Fig. 1 where a team of robots assemble chair parts by attaching them to each other. Once an assembly operation is complete, the partially-assembled structure can be transferred to subsequent assembly operations to be combined with even more parts. We present an example sequence in Fig. 2.

In this paper, we propose an algorithm to plan multi-robot grasping configurations to be used during a sequence of assembly operations. We make some assumptions:

- We assume that our planner is given, as input, the sequence of assembly operations, i.e. the order with which the parts must be assembled. Given the structure of the final assembly, such a sequence can be produced either manually or by using an *assembly planner* (Wilson 1992; Wilson and Latombe 1994; Halperin et al. 2000; Cortes et al. 2008; Knepper et al. 2013). Note that these *assembly planning* algorithms output a sequence only in terms of the assembly parts, not including the robots.
- We assume that local controllers exists to perform the fastening/screwing operations once the parts are brought to the pre-fastening/pre-screwing poses. This paper focuses on the geometric and kinematic problem of grasping the parts at the pre-fastening/pre-screwing poses. The general fine manipulation problem of inserting a peg in a hole under uncertainty is a critical one. While we do not address this problem in this paper, we discuss existing work in this area in Sect. 1.1.

The multi-robot sequential grasping problem imposes a variety of constraints on the robot configurations. Consider

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10514-018-9748-z) contains supplementary material, which is available to authorized users.

✉ Mehmet Dogar
m.r.dogar@leeds.ac.uk

Andrew Spielberg
aespielberg@csail.mit.edu

Stuart Baker
spbaker@csail.mit.edu

Daniela Rus
rus@csail.mit.edu

1 School of Computing, University of Leeds, Leeds LS2 9JT, UK

2 Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

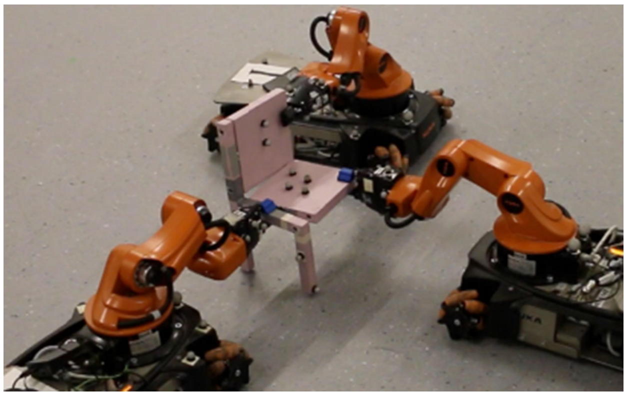
Fig. 1 Three robots at an assembly configuration

the assembly operation scene in Fig. 1. We immediately see one type of constraint: the robot bodies must avoid intersection. In effect, the robots must "share" the free space as each of them grasps a point on the corresponding assembly part and attains a base and arm configuration.

The sequential nature of the task, however, may impose even more constraints. The robots may choose one of two strategies to move a partially-assembled structure from one assembly operation to the next (Fig. 2): (i) The robots can *regrasp*, changing their grasp on the partially-assembled structure, or (ii) they can *transfer* the partially-assembled structure directly to the next operation, keeping the same grasp.

Both strategies have their advantages. If the robots choose transfer, this avoids extra regrasp operations during execution. Regrasps, on the other hand, make the planning problem easier by decoupling operation sequences from each other: In Fig. 2, since the robot commits to transfer the structure between assembly operations 1 and 2, it must plan a grasp of the part which works for both operations. The coupling between multiple operations makes it extremely expensive to solve problems that require long sequences of operations.

Humans use a combination of both strategies during manipulation: we regrasp when we need to, but we are also able to use transfer grasps which work for more than one operation. Given a sequence of assembly operations, how can a team of robots decide when to regrasp and when to transfer? In this paper, we present a planner with this capability. Our planner builds on three key contributions:

- A formulation of this multi-robot planning problem as a discrete constraint satisfaction problem (CSP);
- A planning algorithm for sequences of assembly operations, which simultaneously (i) decides on the assembly operations to connect with direct transfers, and (ii) solves the resulting CSP;
- An algorithm to plan regrasps of assemblies between remaining operations.

CSP formulation

We formulate multi-robot grasp planning as a CSP. In this representation every robot in every assembly operation becomes a variable. Every variable must be assigned a value, i.e. a robot configuration which grasps a particular part. We construct a possible set of values by discretizing robot configurations that grasp the part. We define two types of constraints between variables: *collision constraints* between variables of the same assembly operation; and *transfer constraints* between variables in subsequent operations

We present the details of this CSP formulation in Sect. 3.

Planning for sequences of assembly operations

Ideally, the assembly is transferred from one operation to the other smoothly for all operations in the sequence. This, however, means having transfer constraints between all operations in the CSP. A complete solution requires solving for all the assembly operations at once. In general, complete CSP solvers display exponential worst-case complexity with respect to the number of connected variables (Dechter 2003). Solving the multi-robot grasp planning problem becomes exponentially expensive with increasing number of assembly operations. Similarly, if we can break some of these transfer constraints and create multiple independent CSP problems, we can reduce the planning time exponentially.

Here, we make a key and important assumption: Given two valid grasps of an assembly, it is always possible to *regrasp*, i.e. to transition from one grasp to the other (possibly through a series of intermediate grasps). While one can create overly constrained problems where this may not be possible, in manufacturing and assembly domains, it is safe to assume that robots will be given enough space to perform regrasping operations of assembly parts, e.g. an empty patch of floor space.

Relying on this assumption, our algorithm imposes only a subset of the possible transfer constraints.

When our algorithm imposes a subset of the possible transfer constraints, we recognize that not all transfer constraints have the same level of difficulty; some take more time to solve than others. Our algorithm uses local search techniques for CSPs (Minton et al. 1992) to identify the "easier" transfer constraints and imposes them first. As solutions are found, the algorithm imposes larger and larger sets of transfer constraints.

Our algorithm is an anytime planner: Given more time it generates plans with fewer regrasps and more transfers. The algorithm enables the user to stop the planner and use the current best plan if the time it takes to remove more regrasps from the plan exceeds a certain threshold (e.g. the time it takes to plan and execute those regrasps).

An example output of this algorithm can be seen in the top row of Fig. 2.

We present the details of this algorithm in Sect. 4.

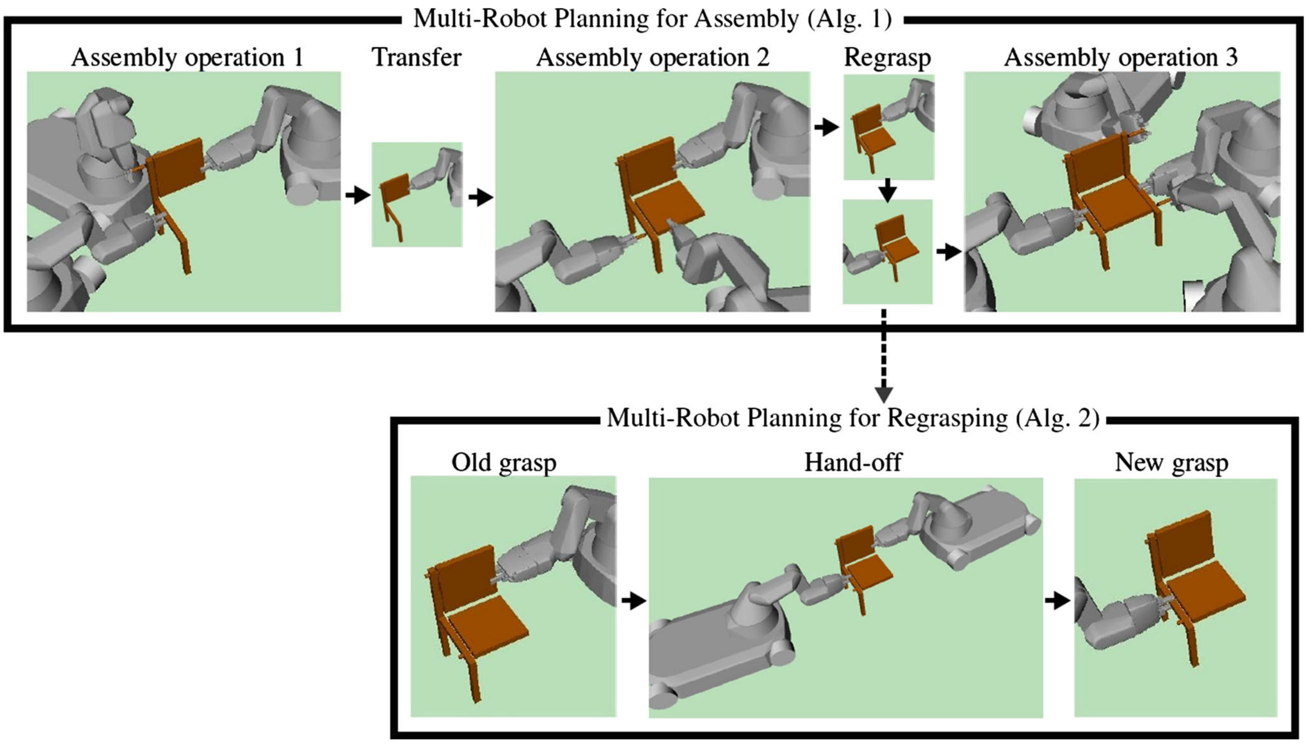

Fig. 2 A sequence of collaborative assembly operations to build a chair. Top row: One robot maintains its grasp after assembly operation 1, transfering the partial-assembly directly to assembly operation 2. Maintaining a grasp is more difficult between assembly operations 2 and 3, due to geometrical constraints. Our algorithm detects this and a regrasp is performed on the partial-assembly. Bottom row: The regrasp is planned as a collaborative hand-off

Planning regrasps

The algorithm outputs a plan where some operations are not connected with a direct transfer. For these operations a regrasp must be planned. In this paper we present an additional algorithm to plan such regrasps. We show how to plan a regrasp also as a multi-robot operation, in which one of the robots of the former operation hands off the assembly to another robot. The new robot grasps the assembly in the way required for the latter operation. We model the hand-off as a multi-robot CSP and solve it. An example output of this algorithm can be seen in the bottom row of Fig. 2.

We present the details of the regrasp planning algorithm in Sect. 5.

This paper is a significantly extended and improved version our work on multi-robot grasp planning (Dogar et al. 2015b). This version presents a completely novel re-grasp planning algorithm (Sect. 5) and a new set of experiments (Sect. 6). We also increased the number and variety of assembly tasks we used in our experiments. We improved and simplified the presentation of the CSP formulation and Algorithm 1.

1.1 Related work

This work is related to two areas that have been traditionally studied separately: *multi-robot manipulation* and *manipulation for assembly*.

In *multi-robot manipulation*, the goal is moving an object from an initial location to a goal location in an environment with obstacles. Multiple robots may need to collaborate for such a task because the reachable space of any individual robot may be limited (e.g. when the robot arms are bolted to the ground) or because an individual robot may not be able to support the object in a stable manner. In each case, feasible robot configurations and motion plans must be found where multiple robots simultaneously grasp the object. The object can also be re-grasped, either by placing it stably on a surface and grasping it again, or by a direct hand-off between robots. Recently there have been important progress in mapping out the manipulation graph (Siméon et al. 2004) of this problem. Harada et al. (2014) developed the manipulation graph for two arms. Dobson and Bekris (2015) then generalized this to multiple arms and proposed methods to deal with the complexity of the resulting problem. Building on earlier work

(Koga and Latombe 1994; Gharbi et al. 2009), these methods provide a general framework for planning multi-robot manipulation operations. In another line of work, Vahrenkamp et al. (2010) propose a sampling-based algorithm which can simultaneously plan feasible and stable multi-robot grasps of an object and the motion of the robots to attain these grasps.

In addition to multi-robot manipulation, our problem is also related to work in *manipulation for assembly*, during which a robot assembles multiple parts together at specific relative poses sequentially (Wan et al. 2015; Wan and Harada 2016). These algorithms use *re-grasp graphs* to identify the sequence of grasps of an object to bring it into the desired grasp. When compared with ours, an important advantage of this line of work is the ability to use a support surface (such as a table or a floor) to place objects in stable configurations and grasp them again. Instead, our approach focuses on *multi-robot* planning of operations. In our approach, some robots hold the partial assembly for other robots to add more assembly parts or simply to re-grasp. This can be necessary, because a stable placement of the partial assembly on a support surface cannot always be found such that the next assembly operation is geometrically feasible. Multi-robot planning also enables robots to assemble multiple parts simultaneously, instead of assembling them one by one.

Our approach takes future assembly operations into account by treating them as constraints during planning. Therefore, it is also related to work in grasp planning that take into account task constraints (Berenson and Srinivasa 2008; Berenson et al. 2011; Dang and Allen 2012). Unlike previous work, however, we focus on planning such grasps in a sequential *and* multi-robot context. This results in additional constraints and computationally expensive problems due to the potentially high number of sequential steps in an assembly task. Our algorithms are tailored to solve exactly this problem: how to deal with constraints that arise due to sequential and multi-robot operations. A particular strategy we use is the progressive application of constraints which is similar to the approaches in Ferbach and Barraquand (1997) and Bayazit et al. (2005).

The effectiveness and necessity of regrasping during manipulation have been recognized for a long time (Lozano-Pérez et al. 1987; Siméon et al. 2004). We show that, by assuming feasibility of regrasps, we can simplify the CSP solutions of manipulation plans significantly. Structures similar to the *grasp-placement space* (Tournassoud et al. 1987) or the *grasp-graph* (Dafle et al. 2014) can be precomputed to satisfy our regrasp feasibility assumption.

Our algorithm takes as input a sequence of relative poses of assembly parts. A seminal problem in robotics is *assembly planning* which addresses the problem of finding such sequences given the final assembled structure and its parts. Efficient algorithms have been developed to address this problem (Wilson 1992; Wilson and Latombe 1994; Halperin et al. 2000; Cortes et al. 2008; Knepper et al. 2013). However, assembly planning algorithms do not represent robots; instead they solve the problem only in terms of the motion and configuration of the assembly parts. In this paper we focus on planning the robot grasps and configurations for an assembly plan. We assume we are given the assembly plan, i.e. a sequence of parts to be assembled.

Our problem can also be thought as an instance of combined *task and motion planning* (TAMP) (Cambon et al. 2009; Bhatia et al. 2010; Kaelbling and Lozano-Pérez 2011; Kaelbling and Lozano-Pérez 2013; Srivastava et al. 2014; Dellin and Srinivasa 2015). From this point of view, a planner can be developed that takes as input only the final assembly. The planner can then simultaneously generate the assembly plan, the robotic grasps on the parts, and the robots' motion. However, the computational complexity associated with the above TAMP planners make it infeasible to solve the long sequence of multi-robot problems that we attack in this paper; e.g. in Sect. 6 we solve tasks that require 80 distinct steps with 5 robots taking part in each of these steps. While we aim to target such a general solution as future work, in this paper we take a step in that direction by solving the problem for a given high-level assembly plan.

Within the TAMP framework, recent work by Lozano-Pérez and Kaelbling (Lozano-Pérez and Kaelbling 2014) is particularly relevant to our formulation, since it also represents sequential manipulation problems as CSPs. These geometric CSPs are formulated by a higher-level task planner. Their focus is on the interface between the task planner and the CSP formulation, and they propose methods for constructing the CSPs efficiently. The CSPs are then solved by an off-the-shelf solver. We propose an algorithm to solve the CSP itself using multiple robots and domain-specific assumptions, such as feasible regrasps.

During the transfer of a partially-assembled structure from one operation to the next, our planner needs to select a location on the assembly for the robot to grasp on. Some locations may be easy to find solutions for, and others may be difficult or even impossible. This is an important problem in *multi-step planning* (Bretl 2006; Hauser et al. 2005) where the planner, considering a constraint among other possible choices, needs to identify how difficult, or impossible, it would be to solve that specific constraint. Our approach, based on representing the problem as a constraint graph, allows us to use the smallest local neighborhood of a constraint in which it can be solved as a measure of how difficult it is to solve the constraint. This enables us to identify and solve the easier constraints first.

We use complete and local methods to solve CSPs. There is extensive literature in this area (see Dechter 2003 for a thorough overview). The compact treatment in Russell and Norvig (2003) covers all the CSP methods we use.

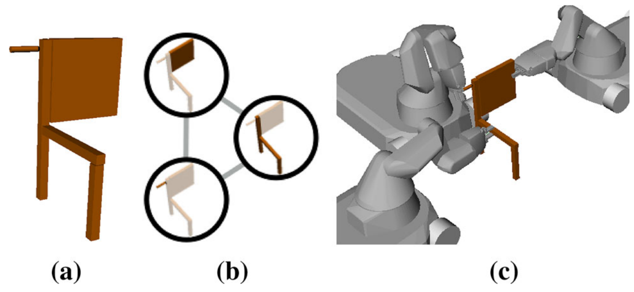

Fig. 3 **a** An example assembly operation to bring three parts together: the side of a chair, the back of a chair, and a fastener. **b** The constraint graph for the assembly operation where each edge represents collision constraints between grasping robots. **c** A multi-robot grasp configuration for the assembly operation

In this work, we assume that local controllers exist to perform insertion/fastening operations once the assembly parts are brought to the pre-insertion poses. This is, however, a critical and non-trivial problem in fine manipulation under uncertainty (Lozano-Perez et al. 1984). In previous work, we used a compliant shape for the fastener tip, such that the fastener is guided by the hole during insertion to deal with such uncertainty (Dogar et al. 2015).

2 Problem

An assembly is a collection of parts at specific relative poses. A simple part by itself is also a (trivial) assembly. Robots perform an *assembly operation*, $o = (\mathbf{A_{in}}, a_{out}, p)$, to produce an output assembly a_{out} from a set of input assemblies $\mathbf{A_{in}}$. We also assume that a three-dimensional pose in the environment, p, is specified as the location of an operation.

We present an example assembly operation in Fig. 3a.

During an assembly operation, input assemblies $\mathbf{A_{in}}$ must be grasped and supported by robots at their respective poses in a_{out} at operation pose p. We assume that a local controller exists to perform the fastening/screwing, once the parts are at the poses specified by the assembly operation.

We do not assume that the robots start with the parts grasped in their hands. Instead, we assume that, initially, the parts are positioned somewhere in the environment (e.g. on the floor or on a desk) and the robots have to find a feasible initial grasp to pick them up, avoiding any environmental collisions at this initial pose of the part (e.g. the floor or the desk). For example, each of the three parts in Fig. 3a-a (the side of the chair, the back of the chair, and the fastener) must be picked up from such initial locations. Our definition of an assembly operation also applies to this initial grasp of a single part a, where $\mathbf{A_{in}} = \{a\}$ is a singleton, $a_{out} = a$, and p is the initial pose of part a in the environment.

A robot can grasp an assembly by placing its gripper at certain poses on the assembly. We assume we can compute a set of such poses, *grasps*, for each assembly a. We illustrate example grasps for simple parts in Fig. 4. We use $\mathbf{Q}$ to represent the robot configuration space, which includes base pose and arm joint configurations. If a configuration $q \in \mathbf{Q}$ places the robot gripper at a grasping pose for assembly a during operation o, we say that "q is grasping a during o". The robots must avoid collisions during assembly operations.

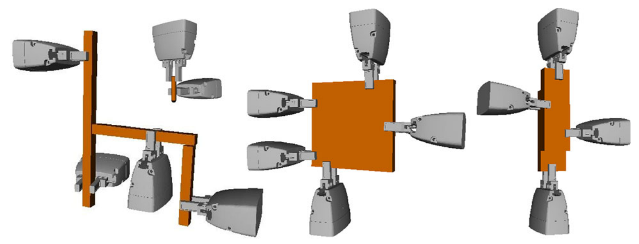

Fig. 4 Example grasps for assembly parts

Robots perform an ordered sequence of assembly operations $\mathbf{O} = [o_i]_{i=1}^{N}$ to gradually build large complex structures: output assemblies of earlier operations are used as inputs in later operations. Robots move the assemblies from one operation to the next.

As an example, we present a sequence of assembly operations to build a chair in Fig. 5a. This example includes eleven operations: three operations in which multiple parts must be assembled, and eight operations where each single part (the leftside of the chair, rightside of the chair, the seat, the back of the chair, and four fasteners) must be grasped at their initial poses. Each arrow indicates an instance where robots move an assembly from one operation to the next.

Given a sequence of assembly operations, we formulate the problem of *multi-robot grasp planning for sequential assembly operations* as finding grasping configurations for all the robots required by the assemblies in all the operations.

2.1 Moving assemblies between operations

Suppose $o = (\mathbf{A_{in}}, a_{out}, p)$ and $o' = (\mathbf{A'_{in}}, a'_{out}, p')$ are two operations such that $a_{out} \in \mathbf{A'_{in}}$; i.e. the output assembly of o is one of the input assemblies of o'. We call o and o' *sequential operations*. a_{out} must be moved to o' after o is completed. There are two ways this can be done: *transfer* and *regrasp*.

To directly *transfer* a_{out}, one of the robots grasping an assembly in $\mathbf{A_{in}}$ can keep its grasp and carry a_{out} to o'. There is flexibility; any $a \in \mathbf{A_{in}}$ may be used for the transfer as long as the grasp is stable for the output assembly. For example, after the assembly operation 1 in Fig. 2, the assembled structure can be transferred either by the grasp on the back of the chair as in the figure, or by the grasp on the side of the chair (i.e. by the robot on the bottom-left corner in assembly operation 1).

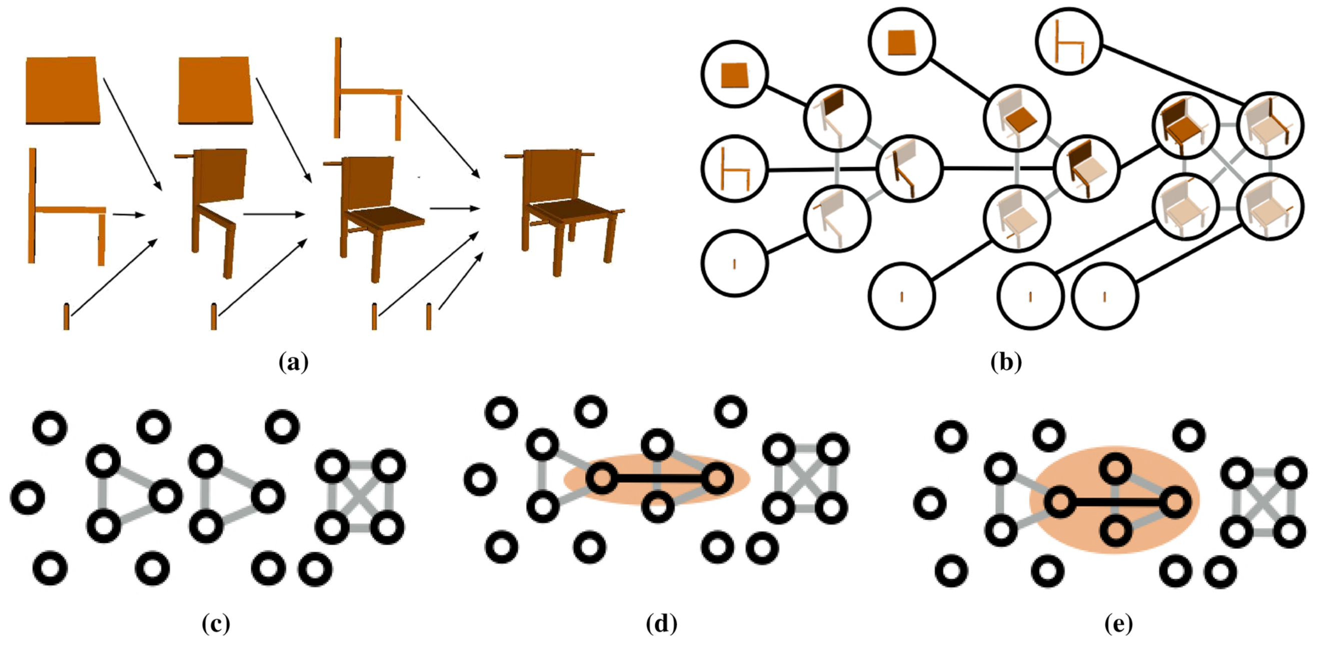

Fig. 5 The chair assembly example. We represent a sequence of assembly operations (**a**) as a constraint graph (**b**). Our algorithm starts by solving the constraint graph with no transfer constraints (**c**). Then it imposes increasing numbers of transfer constraints to the graph and tries to solve them in a small local neighborhood of the graph (**d**). For a certain number of transfer constraints, n, our algorithm iterates over all possible n-combinations of transfer constraints until it can solve one. If none can be solved, our algorithm enlarges the search neighborhood and tries again (**e**). **a** Assembly operations for a chair, **b** A complete constraint graph for the chair, **c** No transfer constraints, **d** Trying to impose one transfer constraint, **e** Searching a larger neighborhood

The alternative to directly transfering is to *regrasp* a_{out}. Robots can regrasp an assembly in different ways: e.g. by first placing it on the floor in a stable configuration and then grasping it again, or with the help of other robots that can temporarily grasp and support the assembly while it is being regrasped. The important implication for our planning problem is that the new grasp of a_{out} can be different from the grasps of all $a \in \mathbf{A_{in}}$. An example is the regrasp after the second assembly operation in Fig. 2.

3 CSP formulation

We formulate multi-robot grasp planning for assembly as a discrete constrained satisfaction problem (CSP).

A CSP is a triple $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, where $\mathbf{X}$ is the set of variables; $\mathbf{D}(x)$ is the domain of variable x, i.e. the discrete set of values that x can be assigned with; and $\mathbf{C}$ is the set of constraints. A solution to the CSP is an assignment of a value to each variable, such that all the constraints are satisfied.

3.1 CSP for a single assembly operation

To plan robot grasping configurations for an assembly operation we can create a CSP and solve it. Below, given an assembly operation $o = (\mathbf{A_{in}}, a_{out}, p)$, we show how to construct the three components of the CSP, $\langle \mathbf{X}_o, \mathbf{D}_o, \mathbf{C}_o \rangle$, representing the planning problem for the operation.

Variables: We create one variable for the grasp of each input assembly of the operation.[1] We use x^a to represent the variable corresponding to the grasp of assembly a. Formally,

$$\mathbf{X}_o = \{x^a \mid a \in \mathbf{A_{in}}\} \tag{1}$$

Domains The domain of the variable x^a is the set of robot configurations grasping the assembly:

$$\mathbf{D}_o(x^a) = \{q \in \mathbf{Q} \mid q \text{ is grasping } a.\} \tag{2}$$

In general there can be a continuous set of robot configurations grasping a, due to redundancy in the kinematics or due to a continuous representation of grasping gripper poses on a part. We discretize this continuous set by sampling uniformly. For each assembly, these grasping configurations must achieve stability and must avoid self-collision and collision with environmental constraints. These checks are performed as the domain for a variable is constructed, and grasp configurations are removed from the domain if they do

[1] If an assembly requires multiple robots to grasp and carry it, e.g. due to its weight, we can create as many variables as there are grasps. For the sake of simplicity, we assume that each assembly can be carried by a single robot in the rest of the paper, but the method can be generalized to multiple grasps.

not give a stable grasp, are in self-collision, or are colliding with environmental obstacles.

Constraints Different types of kinematic and static constraints can be defined for the robot configurations. We define a *collision constraint*, $c(x, x')$, which enforces that two robot configurations assigned to x and x' do not collide. We create a collision constraint between each pair of variables of the assembly operation.

$$\mathbf{C}_o = \{c(x, x') \mid \text{for all } x \text{ and } x' \text{ in } \mathbf{X}, \text{s.t.} x \neq x'.\} \tag{3}$$

Given a CSP, we can represent the variables and constraints in a constraint graph. In this graph, there is a node for each CSP variable, and an edge between two nodes if a constraint exists between the variables. In Fig. 3b we show a constraint graph for the operation shown in Fig. 3a. Each node corresponds to the grasp of a certain part during a certain operation. In the figure, we show the image for the operation inside the node and highlight the image of the part which should be grasped for that variable. The edges between nodes correspond to the collision constraints.

The solution to this CSP gives us non-colliding robot grasping configurations for the assembly operation. An example can be seen in Fig. 3c.

We are, however, further interested in finding such configurations for a sequence of assembly operations.

3.2 CSP for a sequence of assembly operations

Given an ordered sequence of assembly operations $\mathbf{O} = [o_i]_{i=1}^N$, we can create a planning problem by combining the CSPs of each operation into one big CSP. Solving this CSP will be equivalent to solving the CSP of each operation one by one, and will give us non-colliding robot configurations for each operation. This solution, however, will not work if we want robots to directly transfer the assemblies between operations.

We can account for direct transfers by imposing additional constraints between variables of sequential operations. Given two sequential assembly operations $o = (\mathbf{A_{in}}, a_{out}, p)$, $o' = (\mathbf{A'_{in}}, a'_{out}, p')$ such that $a_{out} \in \mathbf{A'_{in}}$, and an assembly $a \in \mathbf{A_{in}}$, we can create a *transfer constraint* between two variables $t({}^o x^a, {}^{o'} x^{a_{out}})$, where ${}^o x^a$ refers to the CSP variable in operation o, and ${}^{o'} x^{a_{out}}$ refers to the CSP variable in operation o'. A transfer constraint enforces that robot configurations assigned to ${}^o x^a$ and ${}^{o'} x^{a_{out}}$ grasp the same part while placing the robot gripper at the same pose on the part. If the CSP with this constraint has a solution, then the assembly a_{out} can be transferred directly from o to o' using the grasp on a. We can also choose not to impose any transfer constraints between o and o'. The robots will then need to perform a regrasp between these operations.

Therefore, given a sequence of assembly operations, $\mathbf{O} = [o_i]_{i=1}^N$, and a set of transfer constraints we would like to impose, $\mathbf{T}$, we can construct a combined CSP, $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, for the planning problem:

$$\begin{aligned} \mathbf{X} &= \cup_{i=1}^N \mathbf{X}_{o_i} \\ \mathbf{D} &= \cup_{i=1}^N \mathbf{D}_{o_i} \\ \mathbf{C} &= \mathbf{T} \cup \cup_{i=1}^N \mathbf{C}_{o_i} \end{aligned} \tag{4}$$

In Fig. 5b we show a constraint graph for the chair assembly operation sequence. Light gray edges correspond to collision constraints, and dark edges correspond to transfer constraints. In this graph all operations are connected with transfer constraints: If we can find a solution, the robots will not need to perform any regrasps.

3.3 Solving a CSP

Having formulated the multi-robot grasp planning problem for sequential assembly operations as a CSP, we can use existing CSP algorithms to solve it. Backtracking search is a widely used and *complete* algorithm for solving CSPs. It searches forward by assigning values to variables such that all assignments obey the constraints. If at any point the algorithm cannot find a value for a variable which obeys the constraints, it backtracks by undoing the most recent assignment. The search continues until an assignment is found for all variables. If there is no solution, backtracking search tries all combinations of value assignments. The worst-case time complexity of naive backtracking search is exponential in the number of CSP variables. One can use domain-independent heuristics to prune the search space. *Minimum remaining value* and *forward-checking* (Russell and Norvig 2003) are two widely used heuristics. Using these heuristics, in the worst-case, bactracking search time complexity is exponential in the number of variables in the largest connected part of the constraint graph.

In our problem, if we impose transfer constraints between all operations, we end up with a fully connected constraint graph. Therefore, in the next section, we propose an algorithm which imposes only a subset of the possible transfer constraints.

Another approach to solving CSPs is by focusing on a local neighborhood of the constraint graph so that the computation time is not affected by the total size of the graph. These local techniques start with an initial assignment of values to variables, identify the conflict regions in the constraint graph, and try to resolve the conflicts only in the local neighborhood of the conflicts. One can use different methods in the local neighborhood, e.g. a complete method like the backtracking search or a heuristic-based search like the *min-conflicts* (Minton et al. 1992) algorithm which greedily minimizes the number of conflicts in the graph. For the *min-conflicts* algorithm, a local neighborhood is enforced usually by limiting the maximum number of steps the algorithm is allowed to run before giving up.

4 Algorithm

We would like to find solutions which involve a small number of regrasps, since each regrasp in the solution will require extra time to plan and execute.

A naive way to find solutions with minimum number of regrasps would be to create transfer constraints between all operations and try to solve the resulting CSP (e.g. the graph in Fig. 5b) with an algorithm such as backtracking search. If this succeeds we have found a solution with no regrasps. If it fails, we can remove one of the transfer constraints and try to solve the resulting CSP problem again to find a solution with one regrasp. If this fails, we can try removing a different transfer constraint, and if that fails, we can try removing two transfer constraints to find a solution with two regrasps; and so on. We call this the *naive CSP solution*.

The problem with the naive CSP solution is that it tries to solve the most difficult problems first: The CSP graph where operations are connected with transfer constraints make the search space exponentially larger. This approach quickly becomes infeasible (see Table 1 in Sect. 6), requiring hours to solve problems with only a few operations.

Instead, we propose an algorithm (Algorithm 1) which first solves the easiest problem, the constraint graph with no transfer constraints, and then tries to improve the solution by imposing an increasing number of transfer constraints as more time is given.

This approach has two advantages. First, it leads to an *any-time planner* which produces a solution quickly and improves it as more time is given. Second, and more importantly, this approach enables us to search in the space of combinations of transfer constraints, identify the combinations that are easier to solve, and solve them first. This cherry-picking of transfer constraint combinations helps us maximize the number of transfer constraints solved in the time allocated to the any-time planner.

4.1 Generating the plan with no direct transfers

We first assume no transfer constraints between operations. Collision constraints remain, but they only constrain variables within an operation. In Fig. 5c, we show this graph for the chair assembly example.

We construct the corresponding CSP and solve it using a complete CSP solver (lines 1–2 in Algorithm 1). Since the constraint graph is divided into N connected components (where each connected component corresponds to one

Algorithm 1 Multi-Robot Grasp Planning for Assembly

```
Input: $\mathbf{O} = [o_i]_{i=1}^N$ is a sequence of assembly operations.
1: $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle \leftarrow$ ConstructCSP($\mathbf{O}$)    ▷ Using Eq. 1-4 with $\mathbf{T} = \emptyset$.
2: $initial\_assignment \leftarrow$ COMPLETESEARCH($\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$)
3: for $n = 1$ to maximum number of transfer constraints do
4:     for enlarging constraint-graph neighborhood $h$ do
5:         for all $\mathbf{T}$ in the set of $n$-combinations of transfer constraints do
6:             $sol \leftarrow$ LOCALSEARCH($\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \cup \mathbf{T} \rangle$, $initial\_assignment$, $h$)
7:             if $sol$ exists then
8:                 $current\_best \leftarrow sol$    ▷ $current\_best$ is the best solution so far. The planner can be stopped 'any-time'.
9:                 Continue at line 3
```

assembly operation), solving the complete problem at this point is equivalent to solving a CSP for each operation separately, and is therefore fast. Any complete CSP solver can be used. We use a custom implementation of backtracking search with minimum remaining value heuristic and forward checking.

The solution on line 2 provides us with an initial assignment of values to variables. At this point we already have a plan, but it is inefficient since executing the plan requires each sequential operation to be interleaved with regrasps.

4.2 Imposing transfer constraints

Once the initial assignment is found, our algorithm starts imposing a gradually increasing number of transfer constraints (line 3) to reduce the number of regrasps. Figure 5d shows one example transfer constraint added to the graph. Between lines 4–9, the algorithm attempts to solve n transfer constraints. If a solution is found, it is recorded as the current best solution (line 8), and the algorithm progresses to $n + 1$ transfer constraints (line 9). One can stop the algorithm anytime and use the current best solution.

The algorithm tries to solve for n transfer constraints as quickly as possible. The algorithm iterates over all valid n-combinations of transfer constraints (line 5). This enables the algorithm to attempt to solve for a different set of n transfer constraints in each iteration, instead of us specifying a particular ordering of the constraints. However, some combinations of transfer constraints can be more difficult to solve for compared to others. Therefore, instead of using a complete search and losing time on difficult combinations, our algorithm performs a local search (line 6) which succeeds or fails quickly in a small neighborhood of the constraint-graph. Local search neighborhood size starts small (Fig. 5d). But, if no solution is found after trying all n-combinations of transfer constraints, the neighborhood gets larger (Fig. 5e) (line 4 in Algorithm 1). Local search variables can be initialized with values using the initial assignment (line 6) or the current best solution. In our implementation we use the current best solution as a heuristic to speed up convergence.

When the algorithm is stopped, the current best solution may require the robots to perform regrasping between certain assembly operations. We present a planner for regrasping in Sect. 5.

4.3 Analysis

We discuss several important properties of our algorithm.

4.3.1 Completeness

Since it relies on a discretization of the robots' configuration spaces, it can easily be shown that Algorithm 1 is resolution-complete.

Proposition 1 *Algorithm 1 is resolution-complete, i.e. it can find a set of valid grasping configurations if such a set exists.*

Proof We use a discrete CSP representation which requires the discretization of the robot configuration space. Assume we are given a resolution with which to discretize. If the algorithm is unable to find a solution with no transfers (as computed on line 2), then the only constraints that the algorithm is unable to satisfy must be those within assembly operations (i.e. collision constraints). This implies one of the following: either the input problem itself is infeasible, or no solution exists at the given resolution of discretization. At a high enough sampling resolution, the second problem disappears. □

In practice, however, it is difficult to implement a resolution-complete version of our planner. The size of the discretized configuration space grows exponentially with the degrees-of-freedom of the robots. Therefore, a discretization with a very high resolution quickly becomes infeasible.

Moreover, the completeness with respect to the grasping configurations does not imply completeness with respect to the complete motion planning problem. First, re-grasps may not be feasible between sequential robot configurations with diffrent grasps. Second, collision-free motion may not be feasible between sequential robot configurations during a transfer.

4.3.2 Optimality

We define optimality as returning the solution requiring the minimum number of regrasps. We do not necessarily aim for optimality: if the time required to remove more regrasps from the plan is more than the time required to execute those regrasping operations, we would like to stop planning and start execution. For this reason, in our implementation we use the greedy *min-conflicts* algorithm for our local search. In practice we have found it to produce good results, however, min-conflicts does not guarantee optimality and may get stuck in local minima.

Algorithm 1, nevertheless, can be an optimal planner if a complete algorithm, e.g. backtracking search, is used to search the local neighborhood (the LOCALSEARCH function on line 6).

Proposition 2 *If a complete local search is used, then Algorithm 1 returns the minimum regrasp solution.*

Proof Our algorithm can terminate early with a suboptimal solution only when it cannot improve the solution via local search for a given number of constraints. However, our algorithm will expand the local neighborhood to include the entire constraint-graph before failing (line 4). If the local search is complete, then this becomes a complete search of the constraint-graph, and a complete search must always find an improvement if it exists. The algorithm cannot terminate if it has not found an optimal solution, and thus it will always return the optimal solution. □

4.3.3 Complexity

The naive CSP solution has an exponential worst-case runtime $\mathcal{O}(\exp(nm))$, where n is the maximum number of robots involved in assembly operations and m is the number of assembly operations (nm is the total number of CSP variables). By comparison, our algorithm's initial solution has the worst-case runtime $\mathcal{O}(m \exp(n))$—exponential in the number of robots per assembly operation but linear in the number of assembly operations (since each operation can be solved independently). Since n is typically small in practice, finding initial solutions is quick. The complexity associated with improving the initial solution depends on the local search technique used. If a complete method such as backtracking search is used, the complexity of improving the solution will approach the complexity of the naive CSP algorithm as more transfer constraints are imposed. We have, however, found the *min-conflicts* greedy search to be a good trade-off between improvement speed and optimality. As we show in Sect. 6 min-conflicts improve the solution quickly and reduces the number of regrasps effectively. This is practical for real-world applications where a small number of regrasps is feasible.

5 Collaborative regrasping

Algorithm 1 can produce plans such that a regrasp is required between operations. It is, however, agnostic to how these re-grasps may be realized. There are many different ways an assembly can be re-grasped. In Sect. 1.1 we present re-grasping approaches that use a surface to place and then pick an object with a new grasp, that use dynamic regrasping to, for example, throw an object into the air and grasp again, and other methods. The re-grasps in the output of Algorithm 1 can be realized using any of these methods.

In this section, we also present one method to plan such regrasps. Particularly, we show how we can plan a regrasp also as a multi-robot collaborative operation.

A regrasp is required between two operations if the robots' grasps of the assembly in the former operation are different from the grasp planned for the latter operation. An example is shown in Fig. 2 top row, between operations 2 and 3. In this case the assembly must be regrasped to achieve the grasp planned for the next operation. We plan such a regrasp as a collaborative operation (Fig. 2 bottom row), in which one of the robots of the former operation hands off the assembly to another robot. The new robot grasps the assembly in the way required for the latter operation.

A hand-off requires both robots to be grasping the assembly simultaneously. Given two sequential operations o and o' which require a regrasp in between, we can model a hand-off as a CSP, $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, similar to the problem defined in Sect. 3:

Variables We create two variables for the hand-off operation, x_{giver} and x_{taker}, corresponding to the two robots taking part in the hand-off:

$$\mathbf{X} = \{x_{giver}, x_{taker}\} \tag{5}$$

Values The set of values for the variable x_{giver} is the set of robot configurations grasping the assembly using one of the grasps in o. We denote this set $\mathbf{Q}_{initial}$. The set of values for the variable x_{taker} is the set of robot configurations grasping the assembly using the grasp required by o'. We denote this set $\mathbf{Q}_{goal}$:

$$\begin{aligned} \mathbf{D}(x_{giver}) &= \mathbf{Q}_{initial} \\ \mathbf{D}(x_{taker}) &= \mathbf{Q}_{goal} \end{aligned} \tag{6}$$

Constraints We impose a collision constraint between the giver and the taker during the hand-off:

$$\mathbf{C} = \{c(x_{giver}, c_{taker})\} \tag{7}$$

This results in a small CSP problem which can be solved directly using a complete search algorithm.

Until now, however, we assumed that a regrasp is possible through a single hand-off. This is usually true, but there may be cases where intermediate grasps are needed. For example if the giver grasp and the taker grasp are very close to each other, it may be impossible to perform a hand-off while avoiding collision. In such a case we fail to solve the CSP problem. We can then construct a new CSP for a hand off with intermediate steps.

Algorithm 2 Multi-Robot Planning for Regrasping

Input: $\mathbf{Q}_{initial}$: The robot configurations corresponding to the initial grasps of the assembly. $\mathbf{Q}_{goal}$: The robot configurations corresponding to the goal grasp of the assembly. $\mathbf{Q}$: All possible robot configurations to grasp the assembly.

```
1:  for $steps = 0$ to MaxNSteps do
2:      Initialize $\mathbf{X}$, $\mathbf{D}$, and $\mathbf{C}$ as empty sets.
3:      for $i = 0$ to $steps$ do
4:          $\mathbf{X} \leftarrow \mathbf{X} \cup \{x_{giver_i}, x_{taker_i}\}$
5:          $\mathbf{C} \leftarrow \mathbf{C} \cup \{c(x_{giver_i}, x_{taker_i})\}$
6:          if $i > 0$ then
7:              $\mathbf{C} \leftarrow \mathbf{C} \cup \{t(x_{taker_{i-1}}, x_{giver_i})\}$
8:      $\mathbf{D}(x_{giver_0}) \leftarrow \mathbf{Q}_{initial}$
9:      $\mathbf{D}(x_{taker_n}) \leftarrow \mathbf{Q}_{goal}$
10:     $\mathbf{D}(x) \leftarrow \mathbf{Q}$ for all $x$ except $x_{giver_0}$ and $x_{taker_n}$
11:     $sol \leftarrow$ COMPLETESEARCH($\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$)
12:     if $sol$ exists then
13:         return $sol$
```

Algorithm 2 is a general algorithm for collaborative regrasping with multiple intermediate steps. The algorithm takes as input the robot configurations corresponding to the initial grasps of the assembly, the robot configurations corresponding to the goal grasps of the assembly, and all possible robot configurations to grasp the assembly at a pre-defined hand-off location in the environment. The algorithm tries to solve the CSP corresponding to a regrasp. If it fails, it increases the number of intermediate steps (line 1) and tries again. For each such attempt a CSP is constructed by creating two variables—giver and taker—for each handoff and imposing collision constraints between them (lines 4–5). The assembly must be transferred between intermediate hand-offs, which are expressed as transfer constraints (line 7). The domain of the first giver and the final taker are quite constrained (lines 8,9), but the robots are free to grasp the assembly however they can during intermediate steps (line 10). We present example regrasping plans in Sect. 6.

6 Experiments and results

We implemented and evaluated our algorithms on a variety of assembly tasks in simulation and using a physical testbed. In this section we present these experiments. We implemented our algorithms in the OpenRAVE environment (Diankov 2010). We used teams of 4 and 5 Kuka YouBot robots[2] for the simulated and physical experiments.

6.1 Comparison to the naive approach

We first present an example task which demonstrates our planner's ability to identify difficult transfer constraints and

[2] http://www.youbot-store.com.

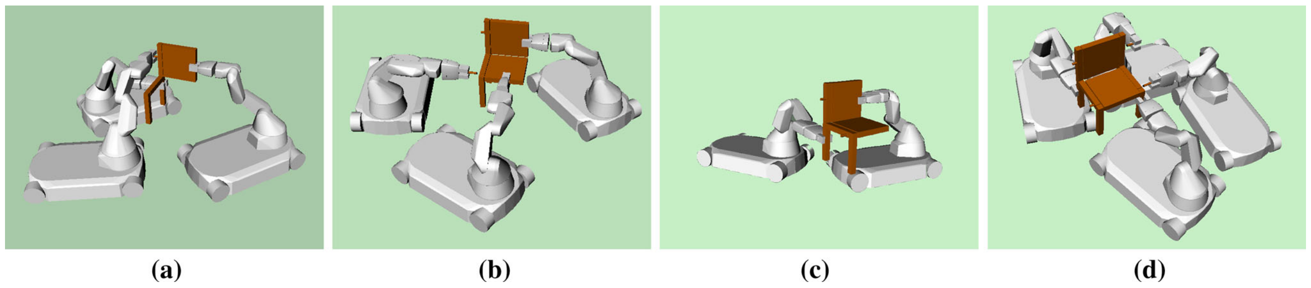
Fig. 6 Key frames of a chair assembly plan. **a**, **b**, **d** are assembly operations, while **c** is a regrasp. Our planner chooses to perform a regrasp right before the highly constrained four-robot operation (**d**)

to plan regrasps instead of solving those constraints. We do this by comparing our planner with the naive CSP planner.

Naive CSP planner This planner, described in Sect. 4, attempts to solve the complete CSP with all the operations connected by transfer constraints. It uses off-the-shelf CSP solver algorithms. In our implementation we use backtracking search.

Our planner Our planner first uses Algorithm 1 to find an increasingly better solution. If the solution cannot be improved (in other words, if no new transfer constraint can be solved) for a certain duration, Algorithm 1 is stopped, and then the remaining sequences of operations are connected through regrasping (Algorithm 2). In our experiments, Algorithm 1 stops if the solution cannot be improved for 15 s. The planning time is the total time it takes to run Algorithms 1 and 2.

We compare these two planners using two versions of the chair assembly task, which we call the *Chair* task and the *Simple Chair* task. The Chair task, presented in Fig. 5a, includes as its last step an operation that requires four robots to collaborate. Planning for four robots simultaneously (for example see Fig. 6d) limits the number of grasps and configurations available to the robots, which creates a highly constrained CSP. The Simple Chair task is a simplified version of the Chair task, where the final step requires only three robots because one of the fasteners is removed from the assembly. The Chair task requires eleven assembly operations (including the initial grasp of eight parts), whereas the Simple Chair task requires ten assembly operations (including the initial grasp of seven parts).

In our experiments we ran each planner 50 times for each task. Each run was given a time limit of 30 min. To construct the CSP problems, we used the exact same sets of domains (uniformly discretized grasps and robot configurations) for the naive CSP planner and our planner. We randomly shuffled the order of these lists between runs to average out a specific order's effect to the planning time.

Both the Naive CSP planner and our planner were able to solve the Simple Chair task within the time limit in all 50 runs. We present the average planning times in the first row of Table 1. Our planner is slightly faster than the Naive CSP for this task.[3] This is due to our algorithm's use of regrasps. The Naive CSP planner solves the complete graph in all 50 runs. In 44 out of 50 cases our algorithm also manages to minimize the number of regrasps to zero. In 6 out of 50 cases, however, Algorithm 1 cannot solve the final transfer constraint for 15 s, is interrupted, and Algorithm 2 kicks in to plan a regrasp. In these 6 cases, the average time Algorithm 2 takes to plan a regrasp is 0.07 s with negligible standard deviation.

Table 1 Planning times averaged over 50 runs

Task	Naive CSP (s)	Our planner (s)
Simple Chair	21.3	15.4
Chair (Fig. 5a)	>711.9	25.6

SD: 27.7 s (Naive CSP on Simple Chair), 11.1 (our planner on Simple Chair), 10.7 (our planner on Chair). We do not have complete data for the Naive CSP planner on the Chair task, since 19 runs timed out and were stopped before generating a plan. Still, the average planning time is guaranteed to be larger than 711.9 s. This is computed by optimistically assuming that these 19 runs were going to generate a plan the moment after they timed out

The advantage of our planner becomes more apparent when we compare the two planners on the more difficult Chair task (second row of Table 1). First, our planner is much faster. Second, the Naive CSP planner runs out of the 30 min time limit in 19 out of the 50 runs, while our planner finds a solution for all 50 runs. In 40 out of 50 runs our planner finds a plan with one regrasp, in 2 runs it finds a plan with two regrasps, and in 8 runs it manages to find a solution with no regrasps. The average time Algorithm 2 takes to plan each regrasp is, again, 0.07 s with negligible standard deviation.

We show the time profile of our algorithm in Fig. 7. The plot is divided into three parts. The first part, shown in red, takes 9.6 s, during which a complete search is performed on the graph without the transfer constraints. This corresponds to the first two lines of Algorithm 1. The transition from the red to the green part marks when the planner has

[3] A two-sample t-test barely rejects ($P = 0.09$) the null hypothesis that our planner is slower than the Naive CSP.

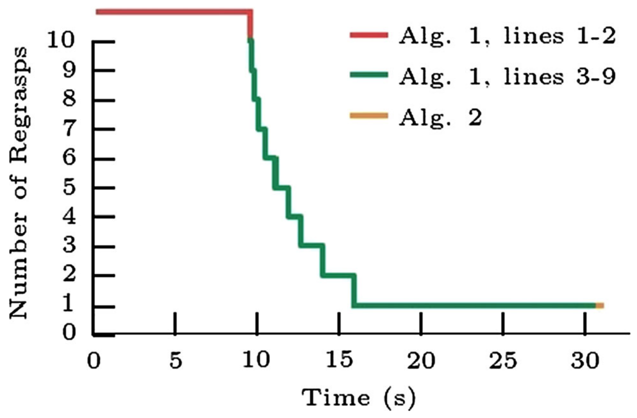

Fig. 7 The time profile of our planner on the chair assembly with 4-robot operations. This plot is drawn by averaging the 40 (out of 50) runs on the Chair task for which our planner finds a solution with one regrasp. The average planning time of these 40 runs were 31 s

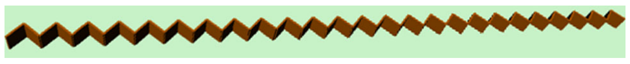

Fig. 8 The Stairs-25 goal assembly with 25 steps, requiring a total of 76 assembly operations connected by 75 transfer constraints

the first solution. At this point, all of the 10 connections between operations are marked as regrasps. Our algorithm starts to impose increasing numbers of transfer constraints (Algorithm 1, lines 3–9). At the beginning the algorithm quickly solves the easy transfer constraints, and the number of regrasps is gradually reduced to 6 within one second. As time progresses, it takes longer to find solutions to transfer constraints. This is not only because the graph becomes more and more connected, but also because specific transfer constraints are harder to solve and our algorithm identifies these constraints (using expanding local search neighborhoods) and postpones their solution. (This can for example be seen in Fig. 6, where the planner chooses to perform a regrasp right before the highly constrained four-robot operation.) When the planner reaches one regrasp, it cannot improve the solution further and after 15 s Algorithm 1 is stopped. Algorithm 2 plans a regrasp for the remaining connection in the graph, which is shown by the small orange extension in the plot.

We present an example plan with a regrasp in Fig. 6. Another example plan was presented in Fig. 2. A video version of a complete plan can be seen in the multimedia extension of this paper.

6.2 Scaling up to long assembly sequences

We performed further tests to show how our algorithm scales with large assembly tasks. We created two different task types. In the first (Fig. 8) the robots attach square parts to each other to create a series of steps. The task instance Stairs-N refers to the case where the robots assemble N steps. Each step takes three robots to assemble. In the second type, shown in Fig. 9, the robots build a grid structure using the same square parts. The task instance Grid-N $\times$ N refers to a $N \times N$ grid. Each grid cell requires five robots working together (except the very first operation), which makes every single operation difficult to plan due to spatial constraints.

We ran the Naive CSP planner and our planner on different instances of these tasks. We ran each planner ten times on each task instance. We gave the planners one hour to run. In this set of experiments, to see our planner's ability to reduce the number of regrasps, we did not stop Algorithm 1 even after it was not able to improve the solution for a long time; we kept it running until the one hour limit was reached. Table 2 summarizes the results. The naive planner returned no solution within one hour for the four larger tasks. Our algorithm, however, was able to return solutions in all cases.

In these experiments we ran Algorithm 1 for an hour to observe its performance. Normally, we would stop it much earlier and still have a good solution. We present an example time profile on the Stairs-16 task in Fig. 10. We see that the number of regrasps is reduced drastically at the very beginning of the one hour. For this task, running Algorithm 1 for only 90 s would reduce the number of required regrasps by half, from 48 to 24.

The Grid tasks are difficult to solve not only because the number of operations are large, but also because each operation is performed by five robots. In these kind of tasks where planning an independent assembly operation is highly constrained by itself, the time to find the initial plan can dominate the planning time. We present an example time profile of our

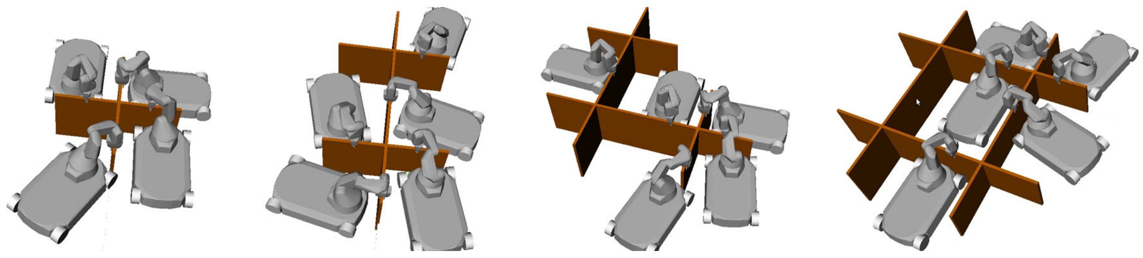

Fig. 9 A plan for a five-robot team performing the Grid-2 $\times$ 2 task

Table 2 Results showing how our algorithm scales with assembly tasks of increasing sizes

Task	Naive CSP	Our planner initial plan (s)	Our planner # transfers
Stairs-9	11.2 s	7.6	27/27 at 24.3 s
Stairs-16	No sol.	81.5	32.9/48 at 1 h
Stairs-25	No sol.	230.4	33.4/75 at 1 h
Grid-2 × 2	24.9	10.9	20/20 at 22.0 s
Grid-3 × 3	No sol.	235.3	40.5/45 at 1 h
Grid-4 × 4	No sol.	1224.6	39/80 at 1 h

The first column shows the planning time of the Naive CSP Planner, where 'No sol.' indicates no solution after one hour of planning. The second column shows the average time it took our planner to produce the initial plan. The third column shows the average number of transfer constraints solved out of the total number of transfer constraints at a certain point in time. For example, for the top row, 27 of the 27 transfer constraints were solved after 24.3 s of planning

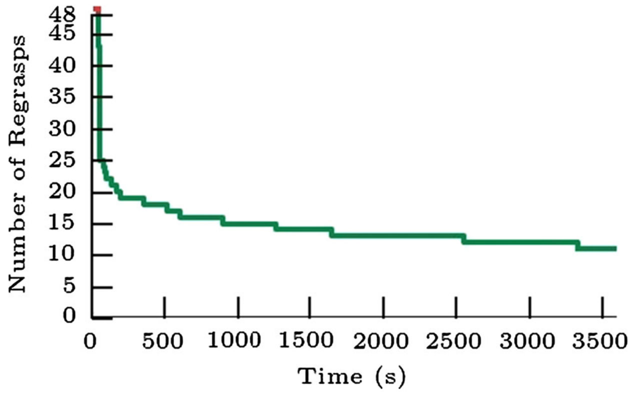

Fig. 10 Time profile of our planner on the Stairs-16 task. This plot is drawn by averaging the three (out of ten) runs for which our planner found a solution with eleven regrasps

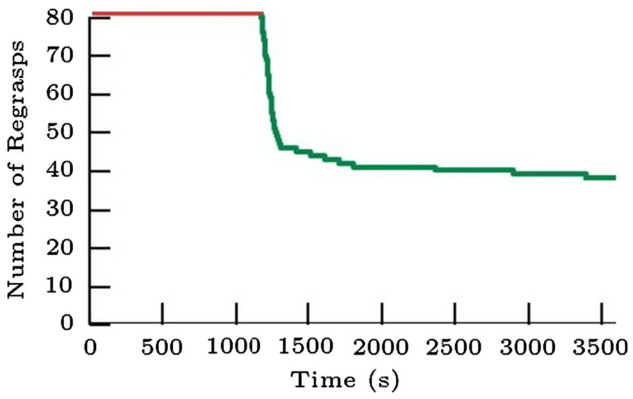

Fig. 11 Time profile of our planner on the Grid-4 × 4 task. This plot is drawn by averaging the eight (out of ten) runs for which our planner found a solution with thirty-eight regrasps

planner in Fig. 11. We can see that it takes significant time to find the initial solution, shown in red in the plot. Once the initial solution is found, almost half of the regrasps are removed in a short amount of time, at which point Algorithm 1 can be stopped and Algorithm 2 can be used to plan regrasps.

6.3 Physical testbed with robots

We are building a real robot team to perform autonomous assembly of complex structures. The algorithm presented in this paper provides our system with the sequences of configurations in which to grasp and assemble parts, enabling fast planning and minimal regrasping operations. In Fig. 12 we present snapshots from the execution of an assembly plan generated by our algorithm. The complete execution can be seen in the video accompanying this paper.

Our system consists of three KUKA Youbot robots, each with an omni-directional base, a 5 degree-of-freedom arm, and a parallel plate gripper. Perception in our system is provided by a motion capture system[4] which is able to detect and track infra-red reflective markers. We localize the robots and the initial location of assembly parts using such markers. We use an RRT planner (Kuffner and LaValle 2000) to move the robots between configurations generated by our algorithm.

We present the initial grasps of three parts in Fig. 12a, b, and Fig. 12c. The robots bring these three parts together in Fig. 12d, using a planned assembly configuration. The robots keep the same grasp on the parts through these operations, enabling them to transfer parts between Fig. 12a–d, b–d,and c–d. In Fig. 12e the remaining part is grasped, and in Fig. 12f the complete assembly of the chair is achieved. Again, the robots keep the same grasp between Fig. 12d–f and Fig. 12e–f. The entire task requires 8 min to complete. This includes execution time as well as planning time for the assembly operations and the motion plans.

While our robots can successfully use the planner output to bring parts to assembly configurations, they need to perform highly precise manipulation operations to actually insert fasteners. Part of our research effort focuses on developing controllers and tools (Dogar et al. 2015) to perform these operations. In this example we use magnets between parts to hold the assembly together.

[4] http://www.vicon.com.

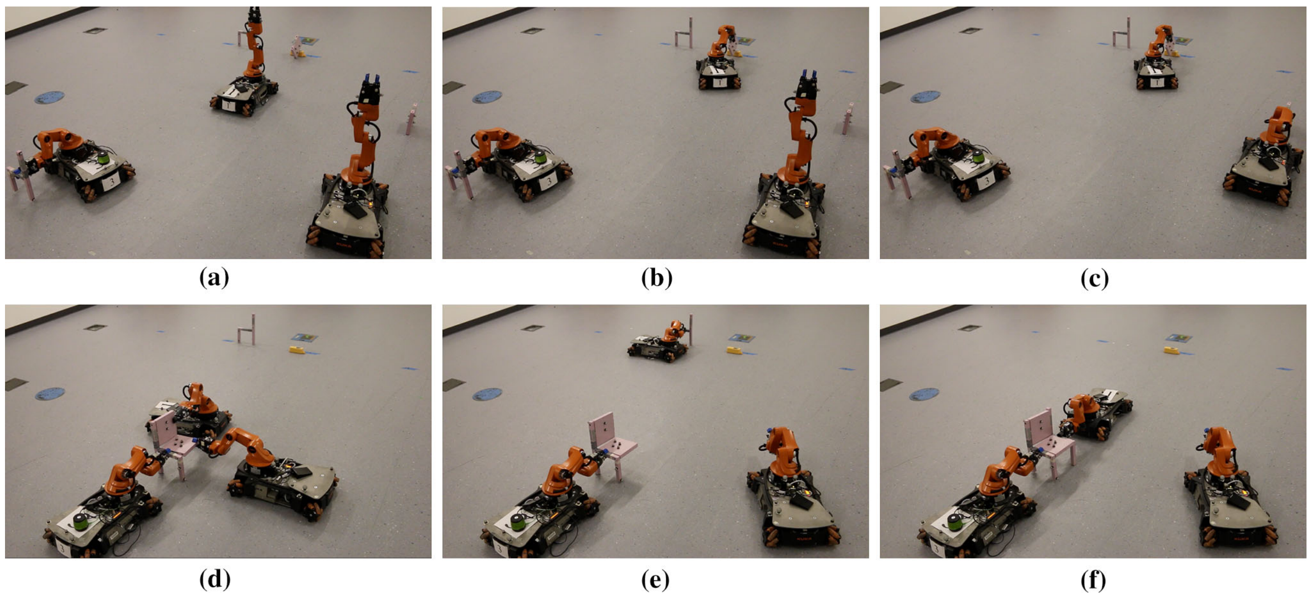
Fig. 12 Multi-robot execution of a chair assembly plan. **a** Grasping right side. **b** Grasping chair back. **c** Grasping chair seat. **d** Assembly of right side, back, and seat. **e** Grasping left side. **f** Assembly of complete chair

7 Discussion

We approach motion planning as post-processing: We first plan the robot configurations and then use a motion planner to connect these configurations to each other. We made this decision since in our domain, where the robots have ample free space, connecting two non-colliding configurations is almost always possible. There were, however, exceptions to this. The last scene in Fig. 9 shows a possible failure mode. While the robot configurations are valid, one of the robots is trapped inside the grid structure.

It is not difficult to integrate motion planning into our planner. In our current formulation, a transfer constraint, as defined in Sect. 3.2, checks if two robot configurations grasp a part at the same point. We can extend this check to include *connectivity*, by running a motion planner between these two configurations. This approach is taken, for example, by Lozano-Pérez and Kaelbling (Lozano-Pérez and Kaelbling 2014) in a cluttered kitchen domain. With this approach, however, each transfer constraint check requires running a motion planner which increase the planning time drastically. In our experience, in domains with ample free space, it is much faster to run the planner without motion planning checks and, in the rare case that the planned configurations cannot be connected, to re-run the planner again by removing the invalid configurations from the CSP variable domains.

One important aspect of robotic manipulation that we did not address in this work is *uncertainty*. During our real robot experiments, uncertainty manifested itself as inaccurate robot grasps, which led to collisions between assembly parts and misalignments. It may be possible to extend our current approach to address some of this uncertainty. This would require planning conservative grasps that would avoid collision even under uncertainty. This will place additional constraints onto the problem and will probably increase the planning times significantly.

Our approach to regrasping focused on hand-offs between robots. This is only one strategy among many that robots can use for regrasping. The robots can also place the object on the ground and then grasp it in new ways. This would require planning for stable placement on the ground, which is by itself an interesting problem especially for irregularly shaped partially-assembled structures.

Finally, in this paper we provide the order of assembly operations as input to our algorithm. The order, in fact, can be computed along with the robot configurations to trade off between minimizing the number of assembly operations and the difficulty of getting too many robots take part in a single operation. This would require combining assembly planning algorithms (Wilson and Latombe 1994) with manipulation planning algorithms such as ours.

Acknowledgements Support for this work has been provided in part by The Boeing Company. Stuart Baker was supported by the MIT Lincoln Labs.

References

Bayazit, O. B., Xie, D., & Amato, N. M. (2005). Iterative relaxation of constraints: A framework for improving automated motion planning. In *2005 IEEE/RSJ international conference on intelligent robots and systems, (IROS 2005)* (pp. 3433–3440). IEEE.

Berenson, D., & Srinivasa, S. S. (2008). Grasp synthesis in cluttered environments for dexterous hands. In *IEEE-RAS international conference on humanoid robots*.

Berenson, D., Srinivasa, S. S., & Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research*, *30*(12), 1435–1460.

Bhatia, A., Kavraki, L. E., & Vardi, M. Y. (2010). Sampling-based motion planning with temporal goals. In *2010 IEEE international conference on robotics and automation (ICRA)* (pp. 2689–2696). IEEE.

Bretl, T. (2006). Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *The International Journal of Robotics Research*, *25*(4), 317–342.

Cambon, S., Alami, R., & Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, *28*(1), 104–126.

Cortes, J., Jaillet, L., & Siméon, T. (2008). Disassembly path planning for complex articulated objects. *IEEE Transactions on Robotics*, *24*(2), 475–481.

Dafle, N., Rodriguez, A., Paolini, R., Tang , B., Srinivasa, S., Erdmann, M., Mason, M., Lundberg, I., Staab, H., & Fuhlbrigge, T. (2014). Extrinsic dexterity: In-hand manipulation with external forces. In *IEEE international conference on robotics and automation*.

Dang, H., & Allen, P. K. (2012). Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *IEEE/RSJ international conference on intelligent robots and systems*.

Dechter, R. (2003). *Constraint processing*. Burlington: Morgan Kaufmann.

Dellin, C. M., & Srinivasa, S. S. (2015). A general technique for fast comprehensive multi-root planning on graphs by coloring vertices and deferring edges. In *2015 IEEE international conference on robotics and automation (ICRA)*.

Diankov, R. (2010). Automated construction of robotic manipulation programs. *PhD Thesis*, CMU, Robotics Institute.

Dobson, A., & Bekris, K. E. (2015). Planning representations and algorithms for prehensile multi-arm manipulation. In *IEEE/RSJ international conference on intelligent robots and systems*.

Dogar, M., Knepper, R. A., Spielberg, A., Choi, C., Christensen, H. I., & Rus, D. (2015a). Multi-scale assembly with robot teams. *The International Journal of Robotics Research*, *34*(13), 1645–1659.

Dogar, M., Spielberg, A., Baker, S., & Rus, D. (2015b). Multi-robot grasp planning for sequential assembly operations. In *IEEE international conference on robotics and automation*

Ferbach, P., & Barraquand, J. (1997). A method of progressive constraints for manipulation planning. *IEEE Transactions on Robotics and Automation*, *13*(4), 473–485.

Gharbi, M., Cortés, J., & Siméon T. (2009). Roadmap composition for multi-arm systems path planning. In *IEEE/RSJ international conference on intelligent robots and systems, 2009. IROS 2009.* (pp. 2471–2476). IEEE.

Halperin, D., Latombe, J. C., & Wilson, R. H. (2000). A general framework for assembly planning: The motion space approach. *Algorithmica*, *26*(3–4), 577–601.

Harada, K., Tsuji, T., & Laumond, J. P. (2014). A manipulation motion planner for dual-arm industrial manipulators. In *IEEE international conference on robotics and automation*.

Hauser, K., Bretl, T., & Latombe, J. C. (2005). Learning-assisted multi-step planning. In *Proceedings of the 2005 IEEE international conference on robotics and automation, 2005. ICRA 2005* (pp. 4575–4580). IEEE.

Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *2011 IEEE international conference on robotics and automation (ICRA)* (pp. 1470–1477). IEEE.

Kaelbling, L. P., & Lozano-Pérez, T. (2013). Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, *32*(9–10), 1194–1227.

Knepper, R. A., Layton, T., Romanishin, J.,& Rus, D. (2013). Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *IEEE international conference on robotics and automation*.

Koga, Y., & Latombe, J. C. (1994). On multi-arm manipulation planning. In *1994 IEEE international conference proceedings of on robotics and automation, 1994* (pp. 945–952). IEEE.

Kuffner, J. J., & LaValle, S. M. (2000). Rrt-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*.

Lozano-Pérez, T., & Kaelbling, L. P. (2014) . A constraint-based method for solving sequential manipulation planning problems. In *IEEE/RSJ international conference on intelligent robots and systems*.

Lozano-Perez, T., Mason, M. T., & Taylor, R. H. (1984). Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, *3*(1), 3–24.

Lozano-Pérez , T., Jones, J., Mazer, E., O'Donnell, P., Grimson, W., Tournassoud, P., & Lanusse, A. (1987). Handey: A robot system that recognizes, plans, and manipulates. In *IEEE international conference on robotics and automation*.

Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, *58*(1), 161–205.

Russell, S. J., & Norvig P. (2003). Artificial Intelligence: A Modern Approach, 2nd edn. Pearson Education.

Siméon, T., Laumond, J. P., Cortés, J., & Sahbani, A. (2004). Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research*, *23*(7–8), 729–746.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)* (pp. 639–646). IEEE.

Tournassoud, P., Lozano-Pérez, T., Mazer, E. (1987). Regrasping. In *IEEE international conference on robotics and automation*

Vahrenkamp, N., Kuhn, E., Asfour, T., & Dillmann, R. (2010). Planning multi-robot grasping motions. In *IEEE-RAS international conference on humanoid robots*.

Wan, W., & Harada, K. (2016). Developing and comparing single-arm and dual-arm regrasp. *IEEE Robotics and Automation Letters*, *1*(1), 243–250.

Wan, W., Mason, M. T., Fukui, R., Kuniyoshi, Y. (2015). Improving regrasp algorithms to analyze the utility of work surfaces in a workcell. In *IEEE international conference on robotics and automation*.

Wilson, R. H. (1992). On geometric assembly planning. *PhD Thesis*, Stanford university: Stanford .

Wilson, R. H., & Latombe, J. C. (1994). Geometric reasoning about mechanical assembly. *Artificial Intelligence*, *71*(2), 371–396.

Mehmet Dogar is a tenure-track University Academic Fellow at the School of Computing, University of Leeds. His research focuses on planning and control for robotic manipulation. Previously he was a Postdoctoral Associate at CSAIL, MIT working on multi-robot planning. He received his Ph.D. from the Robotics Institute, CMU in 2013.

Andrew Spielberg is a Ph.D. student at MIT and a member of the Distributed Robtoics Lab at CSAIL, MIT.

Stuart Baker is a Graduate Student at MIT.

Daniela Rus is the Andrew (1956) and Erna Viterbi Professor of Electrical Engineering and Computer Science and Director of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. She serves as the Director of the Toyota-CSAIL Joint Research Center and is a member of the science advisory board of the Toyota Research Institute. Rus' research interests are in robotics, mobile computing, and data science. Rus is a Class of 2002 MacArthur Fellow, a fellow of ACM, AAAI and IEEE, and a member of the National Academy of Engineering and the American Academy of Arts and Sciences. She is the recipient of the 2017 Engelberger Robotics Award from the Robotics Industries Association. She earned her Ph.D. in Computer Science from Cornell University.