Multi-Robot Grasp Planning for Sequential Assembly Operations

Mehmet Dogar and Andrew Spielberg and Stuart Baker and Daniela Rus

***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 real robot system which uses the output of our planner to grasp and bring parts together in assembly configurations.**

I. INTRODUCTION

We are interested in multi-robot systems which 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 with a fastener. Once an assembly operation is complete, the semi-assembled structure can be transferred to subsequent assembly operations to be combined with even more parts. We present an example sequence in Fig. 2.

This paper addresses the problem of finding robot base and arm configurations which grasp the assembly parts during a sequence of assembly operations.

The problem imposes a variety of constraints on the robot configurations. Take the assembly operation scene in Fig. 1. We immediately see one type of constraint: the robot bodies must not intersect. In effect, they must "share" the free space. The sequential nature of the task, however, may result in even more constraints. A robot may choose one of two strategies to move a semi-assembled structure from one assembly operation to the next (Fig. 2): The robot can *regrasp*, changing its grasp on the semi-assembled structure, or the robot can *transfer* the semi-assembled structure directly to the next operation, keeping the same grasp.

This work was supported by the Boeing Company.

Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA 02139 USA
{mdogar,aespielberg,spbaker,rus}@csail.mit.edu

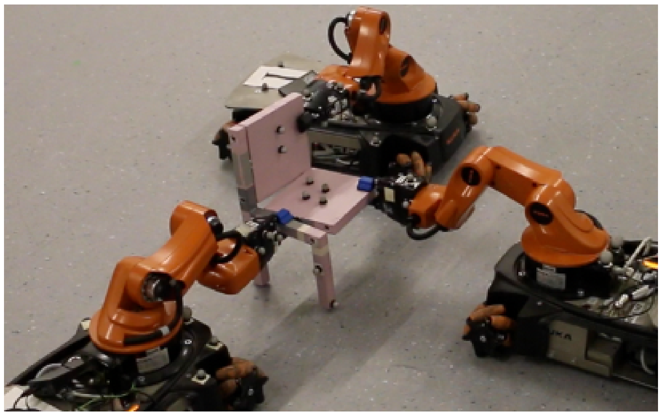


Fig. 1: Three robots at an assembly configuration.

Both strategies have their advantages. If the robot chooses transfer, it avoids extra regrasp operations during execution. Regrasps, on the other hand, make the planning problem easier by decoupling sequence of operations from each other: In Fig. 2, since the robot commits to transfer the structure between assembly operations 1 & 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 with long sequences of assembly 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? We present a planner with this capability: Our algorithm trades off between regrasps and transfers while generating collision-free robot configurations for each assembly operation.

We formulate multi-robot grasp planning as a constraint satisfaction problem (CSP). In this representation every robotic grasp in every assembly operation becomes a variable. Every variable must be assigned a robot configuration which grasps a particular part or semi-assembled structure. We impose two types of constraints: *collision constraints* between variables of the same assembly operation; and *transfer constraints* between variables in subsequent operations.

Ideally, a plan involves no regrasps and the assembly is transferred between operations smoothly. Trying to find a plan with no regrasps, however, means having transfer constraints between all operations. A complete solution requires solving for all the assembly operations at once. In general, complete CSP solvers display exponential complexity with respect to the number of variables [1]. Solving the multi-

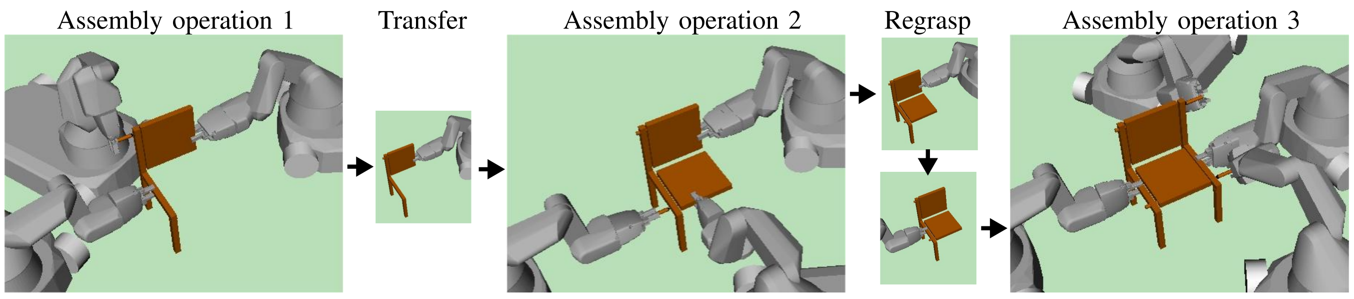


Fig. 2: Sequential assembly operations for of a chair.

robot grasp planning problem then becomes exponentially expensive with increasing number of assembly operations.

Instead, our algorithm starts with a strategy to perform regrasps between all operations. Our key assumption is that, regrasps between any two grasps (possibly through a series of intermediate grasps) are always feasible. This decouples assembly operations from each other. The resulting problem can be solved by solving a small CSP separately for each assembly operation.

After finding this initial solution, our algorithm continues to find solutions with fewer regrasps by imposing sets of transfer constraints. As such solutions are found, the algorithm increases the number of transfer constraints imposed.

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 cost of removing regrasps from the plan exceeds the cost of executing those regrasps.

When imposing a new set of transfer constraints, our algorithm does *not* solve the CSP from scratch: Solutions with fewer (or no) transfer constraints are readily available from previous cycles. We use state-of-the-art local search methods for CSPs, which can be initialized with partial solutions. Local-search methods work only in a locality of the constraint graph and therefore their runtime is not affected by the full size of the CSP [1], leading to fast updates.

A. Related work

Recent work by Lozano-Pérez and Kaelbling [2] also represent 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 CSP formulation, and they propose methods for constructing the CSPs efficiently. The CSPs are solved by an off-the-shelf solver. We propose an algorithm to solve the CSP itself by using domain-specific assumptions, such as feasible regrasps.

The effectiveness and necessity of regrasping during manipulation have been recognized [3, 4]. We show that assuming feasibility of regrasps we can simplify the CSP solutions of manipulation plans significantly. Structures similar to the *grasp-placement space* [5] or the *grasp-graph* [6] can be precomputed to satisfy our regrasp feasibility assumption.

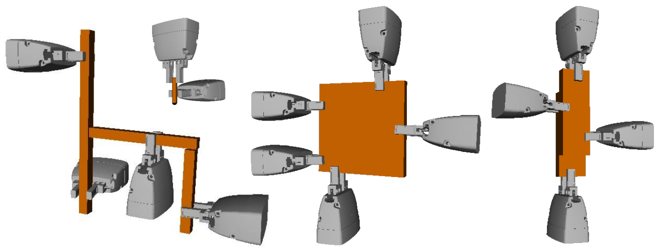


Fig. 3: Example grasps for assembly parts.

Our algorithm takes as input a sequence of relative poses of assembly parts. *Assembly planning* [7, 8] addresses the problem of finding such sequences. In this paper we find *robot configurations* to realize an assembly plan.

Other grasp planners that take into account task constraints [9, 10, 11] and multiple robots [12] exist. Unlike previous work, we focus on planning such grasps in a sequential *and* multi-robot context.

We use complete and local methods to solve CSPs. There is extensive literature in this area but the treatment in Russell and Norvig [1] covers all the methods we use.

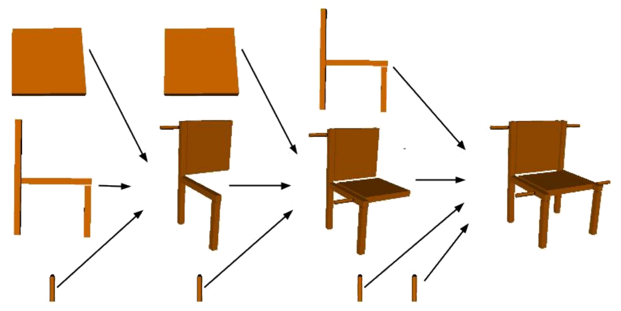
II. Problem

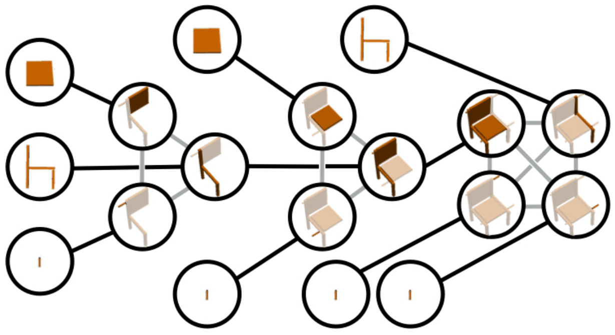
An assembly is a collection of simple parts at specific relative poses. A simple part by itself is also a (trivial) assembly. Robots perform an *assembly operation*, $o = (\mathbf{A_{in}}, a_{out}, p)$, to produce an output assembly a_{out} from a set of input assemblies $\mathbf{A_{in}}$. We also assume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

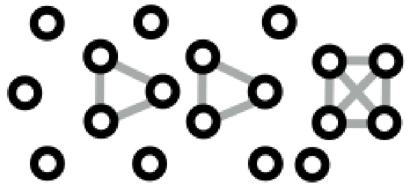
During an assembly operation, input assemblies $\mathbf{A_{in}}$ must be grasped and supported by robots at their respective poses in a_{out} at operation pose p. We assume that a local controller exists to perform the fastening/screwing, once the parts are at the poses specified by the assembly operation.

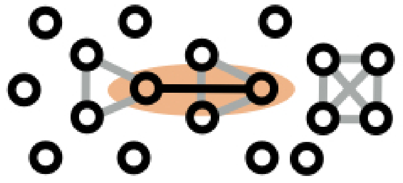
Note that our definition of an assembly operation also applies to the grasp of a single part a, where $\mathbf{A_{in}} = \{a\}$ is a singleton, $a_{out} = a$, and p is the pose of a.

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. 3. We use $\mathbf{Q}$ to represent the robot configuration space, which includes base


(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

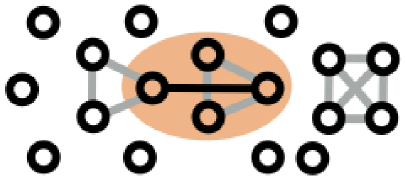

(e) Searching a larger neighborhood

Fig. 4: The chair assembly example.

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 collision during assembly operations.

Robots perform a 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. 4a. This example includes eleven operations: three operations in which multiple parts must be assembled, and eight operations where a single part must be grasped at its initial pose. 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.

A. 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. For example, after the first assembly operation in Fig. 2, the assembled structure can be transferred either by the grasp on the back of the chair as in the figure, or alternatively by the grasp on the side of the chair.

The alternative is to regrasp a_{out} after o is completed. 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 which 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.

III. CSP Formulation

Given a sequence of assembly operations $\mathbf{O}$, we can formulate multi-robot grasp planning as a CSP.

A CSP is defined by a set of variables $\mathbf{X}$, a set of possible values $\mathbf{V}(x)$ that each variable x can be assigned with, and a set of constraints specifying consistent assignments of values to variables. A solution to the CSP is an assignment of values to all the variables that is consistent with all the constraints.

Variables: For our problem, we create one variable for the grasp of each input assembly of each assembly operation. We use ${}^{o}x^{a}$ to represent the variable correponding to the grasp of assembly $a \in \mathbf{A_{in}}$ of operation $o \in \mathbf{O}$.

Values: The set of values for the variable ${}^{o}x^{a}$ is the set of robot configurations grasping the assembly:

$$V({}^{o}x^{a}) = \{q \in \mathbf{Q} \mid q \text{ is grasping } a \text{ during } o.\}$$

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 at a fine resolution.

Constraints: We define two sets of constraints: *collision constraints* and *transfer constraints*. A collision constraint $c(x, x')$ enforces that two robot configurations assigned to x and x' do not collide. We create a collision constraint

$c({}^{o}x^{a}, {}^{o}x^{a'})$ between each pair of variables of the same operation o.

A transfer constraint $t(x, x')$ enforces that robot configurations assigned to x and x' grasp the same part while placing the robot gripper at the same pose on the part.

Given any two sequential assembly operations $o = (\mathbf{A_{in}}, a_{out}, p)$, $o' = (\mathbf{A'_{in}}, a'_{out}, p')$, and an assembly $a \in \mathbf{A_{in}}$, we can create a transfer constraint between two variables $t({}^{o}x^{a}, {}^{o'}x^{a_{out}})$. 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. Each different choice of the transfer assembly $a \in \mathbf{A_{in}}$ corresponds to a different transfer constraint we can impose. Solving for any one of these transfer constraints is sufficient, however.

We can also choose not to create any transfer constraints between o and o'. Our underlying assumption here is that, whatever new grasp is required by a_{out} during o', it will be feasible to achieve it with a regrasp after o — possibly through a number of intermediate grasps. This is a reasonable assumption in our domain where there is ample space in the environment for robots to change from one feasible grasp to another feasible grasp.

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. 4b we show a constraint graph for the chair assembly. 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. 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.

A. Solving a CSP

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 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* [1] are two widely used heuristics.

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* [13] algorithm which greedily minimizes the number of conflicts in the graph. For *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.

IV. 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. 4b) 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. As we will show in the results, this approach quickly becomes infeasible, requiring hours to solve problems with only a few operations.

Instead, we propose an algorithm (Alg. 1) which works in the opposite direction: it 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 *anytime planner* which produces a solution quickly and improves it as more time is given. The planner can be stopped anytime after the initial solution has been achieved and the current solution with the minimum number of regrasps can be used. This, for example, enables the user to stop planning if the planning time spent on imposing new transfer constraints exceeds the time which is required to plan and execute those regrasps. Second, this approach enables the use of local search algorithms to quickly identify easy-to-solve transfer constraints. We would like to solve easy transfer constraints first since we want to minimize the number of regrasps as much as possible before the time allocated to the planner runs out.

A. Generating the “All-Regrasps” Plan

We first assume no transfer constraints between operations. Collision constraints remain, but they only constrain variables within an operation. Hence, the constraint graph is divided into N connected components, where each connected component corresponds to one assembly operation. In Fig. 4c, we show this graph for the chair assembly example.

Algorithm 1 Multi-Robot Grasp Planning for Assembly

```
Input: $\mathbf{O} = [o_i]_{i=1}^N$ is a sequence of assembly operations.
 1: for each $o_i$ in $\mathbf{O}$ do
 2:     $sol[o_i] \leftarrow$ BACKTRACKINGSEARCH($o_i$)
 3: $best \leftarrow \{sol[o_i]\}_{i=1}^N$
 4: for $n = 1$ to MaxTransferConstraints do
 5:     $best \leftarrow$ SOLVETRANSFERCONSTRAINTS($n$,$best$)
 6: procedure SOLVETRANSFERCONSTRAINTS($n$,$seed$)
 7:     for enlarging neighborhood $h$ do
 8:         for each $\mathbf{T}$ in TransferConstCombinations($n$) do
 9:             $sol \leftarrow$ SOLVECSPLOCAL($\mathbf{T}, h, seed$)
10:             if $sol$ exists then
11:                 return $sol$
```

We solve each of these connected components separately using a complete CSP solver (lines 1-2 in Alg. 1). Any complete CSP solver can be used. We use an implementation of backtracking search with minimum remaining value and forward checking.

The collection of solutions of all operations gives one solution for the complete graph, which we treat as the current best solution (line 3). At this point we have a valid plan, but it is inefficient since executing the plan requires each sequential operation to be interleaved with regrasps. We call this solution the "all-regrasps" solution.

B. Imposing Transfer Constraints

Once the "all-regrasps" solution is found, our algorithm starts imposing a gradually increasing number of transfer constraints (lines 4-5) to reduce the number of regrasps. Fig. 4d shows one example transfer constraint added to the graph. The procedure SOLVETRANSFERCONSTRAINTS attempts to solve n transfer constraints. If a solution is found, it is recorded as the new best solution, and the algorithm progresses to $n + 1$ transfer constraints. One can stop the algorithm anytime and use the current best solution.

The procedure SOLVETRANSFERCONSTRAINTS tries to solve for n transfer constraints as quickly as possible. It iterates over all valid n-combinations of transfer constraints (line 8). During this iteration we prioritize combinations which include smaller combinations that we have previously found solutions for. If we cannot find a solution for these prioritized combinations we then try all combinations.

Instead of searching the complete graph and losing time on difficult combinations, our algorithm performs local search (line 9) which succeeds or fails quickly. Local search variables are initialized with values from the current best solution ($seed$). Local search neighborhood size starts small (Fig. 4d) but gets larger (Fig. 4e) if no solution can be found (line 7).

C. Analysis

We analyze several important properties of our algorithm.

1) Completeness:

Proposition 4.1: Algorithm 1 is resolution-complete.

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 in line 3), 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. ■

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.

Alg. 1, nevertheless, can be turned into an optimal planner if a complete algorithm, e.g. backtracking search, is used to search the local neighborhood.

Proposition 4.2: If a complete local search is used, then Algorithm 1 returns the minimum regrasp solution.

Proof: Our algorithm can terminate immaturely with a suboptimal solution only when it cannot improve the solution via local search for a given number of constraints (line 5 in Alg. 1). However, our algorithm will expand the local neighborhood to include the entire graph before failing (line 7). If the local search is complete, then this becomes a complete graph search, and a complete graph 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. ■

3) Complexity: The naive CSP solution has an exponential runtime $\mathcal{O}(\exp(n{*}m))$, where n is the maximum number of robots involved in assembly operations and m is the number of assembly operations ($m * n$ is the total number of CSP variables). By comparison, our algorithm's initial solution has 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 very small in practice, finding initial solutions is generally 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 §V min-conflicts improve the solution quickly and reduces the number of regrasps effectively. This is very practical for real-world applications where a small number of regrasps is feasible.

V. Experiments and Results

We implemented and evaluated our algorithm on an the chair assembly example. We also performed experiments to show that our algorithm can scale up to solve large problems.

A. Chair Assembly

We show the sequence of operations in Fig. 4a. The number of robots required by the complex operations are 3, 3, and 4, respectively. The operations require the semi-assembled structures to be transferred twice and simple parts to be transferred eight times, totaling to ten potential regrasps. We implemented our algorithm and evaluated it in the OpenRAVE environment [14] with four KUKA YouBot robot models [1]. We presented the chair parts to the robots in an environment with some obstacles, presented in Fig. 6a. These obstacles make the problem even more constrained making a no-regrasp solution impossible. Particularly one of the chair side parts must be regrasped after its initial grasp.

We ran our algorithm on the chair example 20 times. In 17 of these runs, our algorithm found the optimal solution with one regrasp and in 3 runs it generated a solution with two regrasps. We plot how our algorithm reduces the number of regrasps with time in Fig. 5. We plot the results for the 17 runs with one-regrasp solutions (red points) separately from the 3 runs with two-regrasps solutions (light green points) since they display different and consistent trends. Each data point marks the average time it took our algorithm to produce a plan with the number of regrasps given on the vertical axis. The horizontal bars show the standard deviations. Our algorithm generates the "all-regrasps" solution in about 4 seconds and then improves the solution every few seconds. The difference between the trends that find one-regrasp solutions (red points) and two-regrasp solutions (light green points) exists because as we impose increasing number of transfer constraints we prioritize combinations which include smaller combinations that we have previously found solutions for, as explained in §IV-B. This, combined with the greedy nature of min-conflicts local search, can create different, possibly non-optimal, trends of solutions for the same problem.

We present part of an example plan in Fig. 6(b)-(d). The robots are able to transfer the assembly between the complex operations without a regrasp: The left-most robot holding onto the side of the chair keeps its grasp fixed and transfers the assembly between all three operations.

We also compare the performance of our algorithm with the naive CSP solution mentioned in §IV. This algorithm is optimal, but also naive in that it tries to solve the full CSP at once. The naive algorithm exceeded the one-hour time limit in 5 of 5 runs. Our algorithm generates plans which include only one or two regrasps in seconds.

Note that we do *not* provide our algorithm with information about the number of tractable transfer constraints to solve. Our algorithm automatically discovers and postpones the solution of intractable or infeasible constraints. In the above example, if given the problematic constraint, one could modify the naive optimal algorithm to ignore that constraint and find a single-regrasp solution. This will lead the naive algorithm to find a quick solution for the single-regrasp case. This modification, however, requires identifying a priori which transfer constraints are difficult to solve. This identification is a challenging problem itself. Our algorithm's power is that it can identify difficult constraints automatically.

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

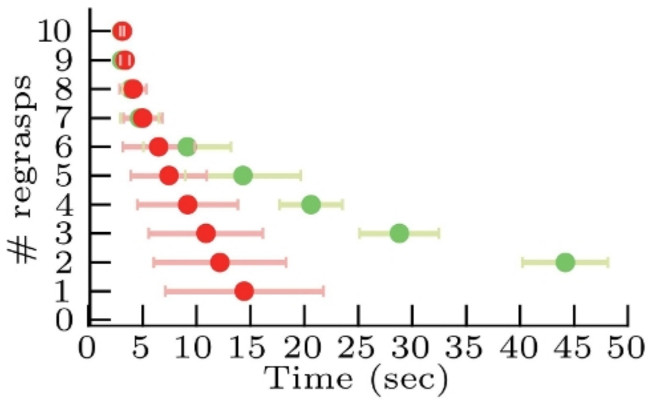


Fig. 5: Time to generate plans with decreasing number of regrasps. In 17 of 20 runs (red points) our algorithm generated a plan with one regrasp. In 3 of 20 runs (light green points) our algorithm generated a plan with two regrasps.

Problem	Avg. # of transfers solved	Avg. time per transfer solution	Naive method planning time
Stairs-4	12 of 12	0.1	1.1
Stairs-9	27 of 27	0.9	11.2
Stairs-16	32.9 of 48	80.7	-
Stairs-25	27.2 of 75	143.4	-
Grid-2x2	20 of 20	1.1	24.9
Grid-3x3	40.5 of 45	98.3	-

TABLE I: Results showing how our algorithm scales with problems of increasing sizes. Times are in seconds.

B. Scalability

We performed further tests to show how our algorithm scales with large assembly problems. We created two different problem types. In the first (Fig. 8) the robots attach square parts to each other to create a series of steps. The problem instance Stairs-N refers to the case where the robots assemble N steps. Each step takes three robot to assemble. In the second type, shown in Fig. 7, the robots build a grid structure using the same square parts. The problem instance Grid-NxN refers to a $N \times N$ grid. Each grid cell requires five robots working together, which makes every single operation very difficult to plan due to spatial constraints.

We ran our planner on different instances of these problems ten times. Tab. I summarizes the results, including the performance of the naive planner for these problems. The naive planner was not able to return a solution within one hour for the larger problems. Our algorithm, however, was able to solve many transfer constraints, as shown in the second column of the table. In the third column we

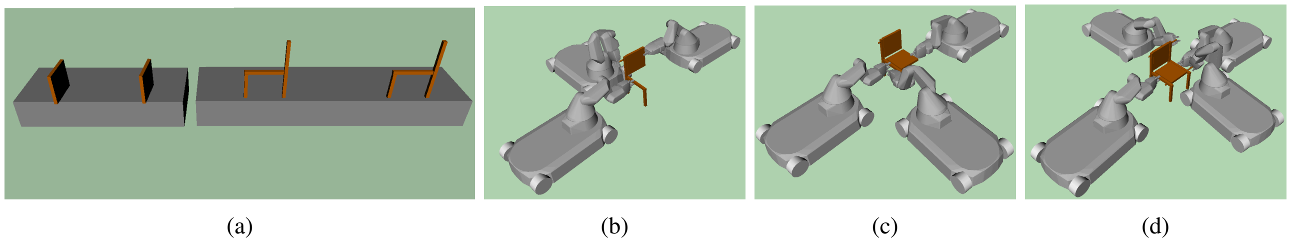


Fig. 6: (a) Initial locations of chair parts. (b)-(d) Solution for the assembly of our chair example.

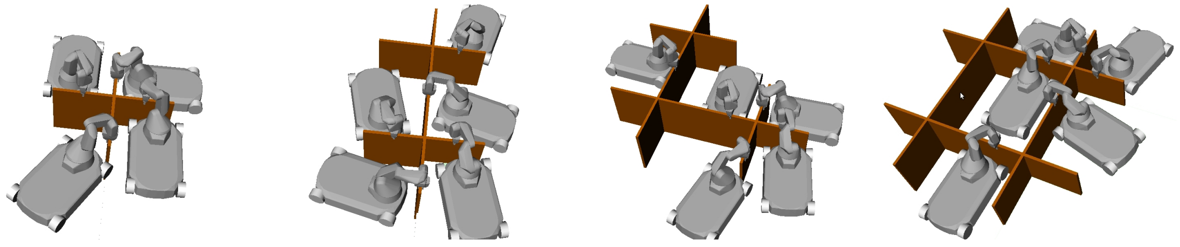


Fig. 7: A plan for a five-robot team building a grid using square blocks.

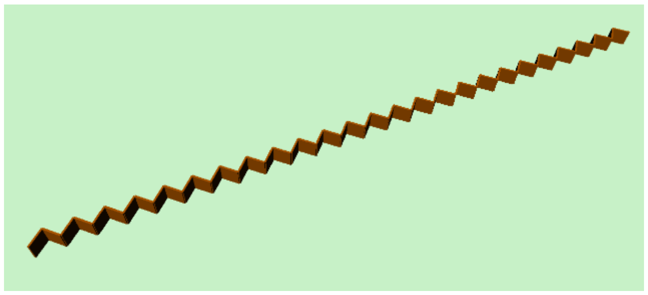


Fig. 8: The goal staircase with 25 assembly operations.

show the average time it took our algorithm to solve one transfer constraint in each instance. Note that earlier transfer constraints are usually solved faster as shown in Fig. 5, and therefore many transfer constraints are solved much faster than the average times shown in Tab. I earlier in the planning.

The last scene in Fig. 7 shows a possible failure mode for our planner. While the robot configurations are valid, one of the robots is trapped inside the structure. As our planner does not check for reachability to and from these configurations, one may expect such problems. A possible solution is to impose a new *reachability constraint* between configurations and use a motion planner to solve them. Lozano-Pérez and Kaelbling [2] propose methods to perform these kinds of checks in a fast way.

C. Real robot implementation

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. 9 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[2] which is able to detect and track infra-red reflective markers. We localize our robots and the initial location of assembly parts using such markers. We use an RRT planner [15] to move the robots between configurations generated by our algorithm.

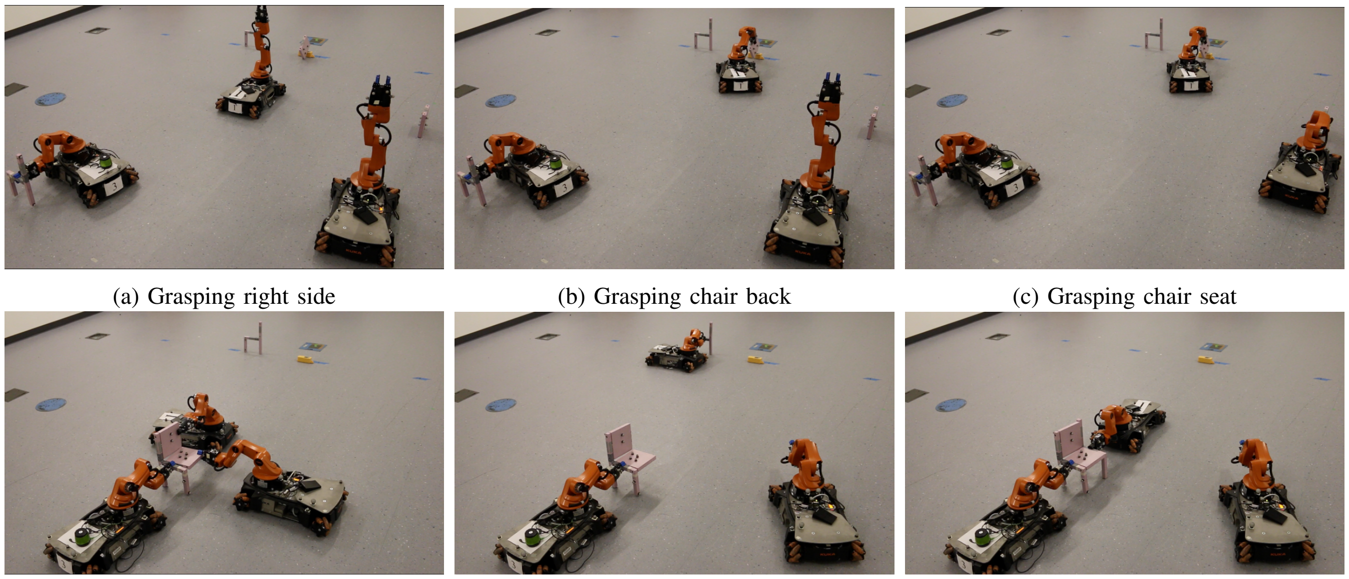
We present the initial grasps of three parts in Fig. 9a, Fig. 9b, and Fig. 9c. The robots bring these three parts together in Fig. 9d, using a planned assembly configuration. The robots keep the same grasp on the parts through these operations, enabling them to transfer parts between Fig. 9a-Fig. 9d, Fig. 9b-Fig. 9d, and Fig. 9c-Fig. 9d. In Fig. 9e the remaining part is grasped, and in Fig. 9f the complete assembly of the chair is achieved. Again, the robots keep the same grasp between Fig. 9d-Fig. 9f and Fig. 9e-Fig. 9f.

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. We are currently developing controllers and tools [16] to perform these operations. In this example we use magnets between parts to hold the assembly together.

VI. Future Work

A system that can go from a design input to complete assemblies requires the development of more advanced techniques both in control and reasoning. For example, error

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



(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

Fig. 9: Multi-robot execution of a chair assembly plan.

accumulated from factors such as loose part grips and localization uncertainty require on-board local controllers for fine manipulation operations. Similarly, a variety of constraints must be taken into account for a robust system, e.g. *stability constraints* which require that the grasps on assembly keeps it stable with respect to gravity and other forces that arise during an assembly operation.

References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed., 2003.

[2] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[3] T. Lozano-Pérez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse, "Handey: A robot system that recognizes, plans, and manipulates," in *IEEE International Conference on Robotics and Automation*, 1987.

[4] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[5] P. Tournassoud, T. Lozano-Pérez, and E. Mazer, "Regrasping," in *IEEE International Conference on Robotics and Automation*, 1987.

[6] N. Dafle, A. Rodriguez, R. Paolini, B. Tang, S. Srinivasa, M. Erdmann, M. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge, "Extrinsic dexterity: In-hand manipulation with external forces," in *IEEE International Conference on Robotics and Automation*, 2014.

[7] R. H. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994.

[8] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *IEEE International Conference on Robotics and Automation*, 2013.

[9] D. Berenson and S. S. Srinivasa, "Grasp synthesis in cluttered environments for dexterous hands," in *IEEE-RAS International Conference on Humanoid Robots*, 2008.

[10] D. Berenson, S. S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[11] H. Dang and P. K. Allen, "Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[12] N. Vahrenkamp, E. Kuhn, T. Asfour, and R. Dillmann, "Planning multi-robot grasping motions," in *IEEE-RAS International Conference on Humanoid Robots*, 2010.

[13] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1, pp. 161–205, 1992.

[14] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, CMU, Robotics Institute, August 2010.

[15] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000.

[16] M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus, "Towards coordinated precision assembly with robot teams," in *International Symposium of Experimental Robotics*, 2014.