Distributed Assembly with AND/OR Graphs

Ross A. Knepper, Dishaan Ahuja, Geoffrey Lalonde, and Daniela Rus

***Abstract*— We describe a planning system in development for efficiently and scalably distributing the task of complex object assembly among multi-robot or human-robot teams. The planner utilizes an AND/OR graph to concisely represent feasible distributed assembly sequences for a given number of available workers (robots or humans). By representing all possible assembly plans in the AND/OR graph, the planner is able to defer commitment to a single plan. This attribute mirrors the natural human approach to collaboration and enables the system to handle failures efficiently.**

I. Introduction

The IkeaBot [5] system autonomously plans and executes assembly procedures using a STRIPS-style [2] symbolic planning language based on primitive actions with pre- and post-conditions, This system generates complex, linear plans for a single tightly-coupled robot team. With more complex assembly problems, it becomes difficult for multiple robots to exploit latent parallelism in the plan because the symbolic planner needlessly overcommits to a total ordering of the plan steps.

There are also a number of functional reasons to delay commitment to an assembly plan. When planning with a human partner, his or her actions are unpredictable, and we would like the robots to adapt to human preferences when reasonable rather than forcing the human to adapt to the robot's arbitrary preference. Additionally, for complex assemblies, the planner typically operates from a position of incomplete knowledge. As the assembly proceeds, new observations may alter the preferred approach. Failures may even occur, which necessitate a new assembly plan. Although IkeaBot has the ability to handle failures [10], it lacks the capability to resolve them by sensibly reordering plan steps.

This paper contributes a distributed planning system that defers selection among all possible assembly sequences for assembly problems such as in Fig. 1. Each robot plans its own next step as part of a multi-worker plan in order to synchronize operations and maximize the productivity of the team.

In this paper, we contribute progress toward the following:

1) a distributed system that plans for the individual as a member of a team
2) practical human-robot collaboration in real time, and
3) efficient failure handling.

*This work was supported by the Boeing Company.

Knepper is with the Department of Computer Science, Cornell University, rak@cs.cornell.edu. The work was performed at MIT.

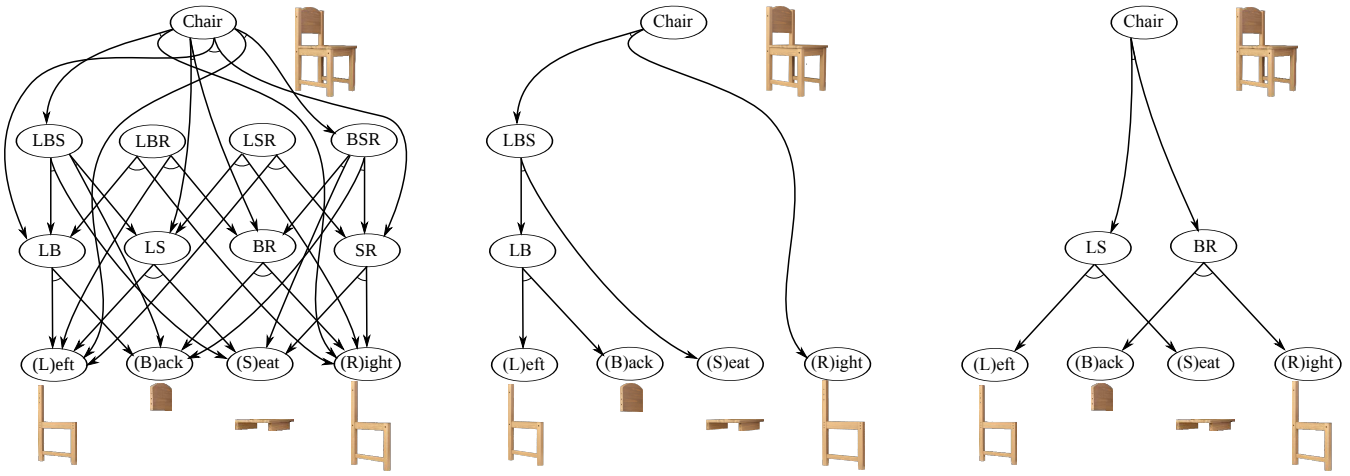
Ahuja, Lalonde, and Rus are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, {dishaan,glalonde,rus}@mit.edu

Fig. 1: We describe a decentralized parallelizing assembly planner for structures such as the IKEA Ekorre rocking moose. Its assembly can be parallelized into as many as five parallel tasks.

II. Assembly Planning

Symbolic planning problems pose start and goal configurations coupled with a set of actions to transform the state of the system. Each action includes a collection of fluents encoding pre- and post-conditions. Such conditions flag state dependencies inherent in performing that action. STRIPS [2] was the first planner and formal language for symbolic planning problems. PDDL [6] standardized symbolic problem specification. Several planners based on PDDL have become popular [3, 7]. We proposed a new object-oriented symbolic problem description language called ABPL [5], which we use for multi-robot assembly planning. In that work, the parallelism of closely coordinated robot action was explicitly encoded within the symbolic actions.

Graph-based approaches, by contrast, attempt to represent the structure of the assembly problem, typically with some details abstracted away. For example, the graph might represent the connectivity of parts in the blueprint [8], the possible partial assembly states, or possible subassemblies. A complete assembly state is a partition of all parts into a



(a) AND/OR graph of the Sundvik chair. (b) Serial plan for one worker. (c) Parallel plan for two workers.

Fig. 2: A simple AND/OR graph and two plans. Note the graph has "dead ends" at *LBR* and *LSR* due to problem constraints.

Fig. 3: An exploded view of the complete disassembly of the Sundvik chair. The AND/OR graph and two alternative assembly plans for this chair are shown in Fig. 2

collection of subassemblies, wherein a single subassembly represents a set of attached parts without regard to the state of the rest of the assembly. The AND/OR graph [1] has been used to represent partial orderings on individual subassemblies. Because AND/OR graphs represent only the part of the state required to describe a subassembly in a node, they are exponentially more space efficient than state graphs. This paper extends the work of de Mello and Sanderson to parallelized multi-agent assembly.

Several efforts have developed centralized planning systems for assembly. Wilson and Latombe [11] describe a geometric approach to assembly planning by virtually disassembling the completed object. The key insight is that it's easier to select a valid sequence by doing a disassembly since dead-ends are impossible and geometric constraints readily identify parts available for removal.

Suchman [9] reveals sociological findings that humans engaged in complex activities, such as assembly, do not typically plan their actions in advance in detail. Instead, they make approximate plans that position themselves correctly to rely on intuition and skill to achieve the task. In fact, humans derive their efficiency from the flexibility afforded by a lack of constraints imposed by a rigid plan. We sought to incorporate this human characteristic into our representation of automated assembly planning in order to make the collaborative assembly planning algorithm adaptable to the unpredictable behavior of the robot's human partners.

III. AND/OR Graph Representation

An AND/OR graph is a directed hypergraph, in which each edge links one parent to multiple children as a child group. We employ this directed acyclic graph (DAG) to describe the composition of the various subassemblies involved in an assembly planning problem. It is thus used to reduce an assembly planning problem into various subproblems, many of which can be solved independently and in parallel.

Each node in the graph represents a possible subassembly. This includes the completely assembled object, as well each base part, which we regard as a trivial subassembly. Consider the example AND/OR graph in Fig. 2a for assembly of a chair (Fig. 3). Each node has one or more parents (except the node representing the completely assembled chair) and zero or more child groups. A node *LBS* is a parent of node *LB* if there exists another node *S* in the graph such that the subassemblies represented by *LB* and *S* can be combined to create the subassembly represented by LBS. In this case, *LB* and *S* are termed *and-siblings* of each other and a child group of *LBS*. Two other subassemblies *LS* and *B* also combine to form the same subassembly *LBS* in another way, so they are called *or-siblings* of *LB* and *S*. By definition, base parts have zero children. This structure defines constraints on the

possible steps that can be taken to assemble an object. Only valid assembly operations are encoded in the AND/OR graph.

A. AND/OR *Graph Generation*

IkeaBot [5] includes a geometric planner that uses annotated CAD data to construct a blueprint of the final structure derived only from the form of the base parts. Encoded within the blueprint is connectivity information among the parts that make up the object. Possible subassemblies can be deduced from this information. The AND/OR graph is generated by combinatorially enumerating pairs of subassemblies and noting the larger subassemblies they produce. We exclude from consideration any join operation that attaches three or more parts simultaneously.

B. AND/OR *Graph Planning Algorithm*

Any of several classic heuristic search algorithms can be used to find an assembly plan from the AND/OR graph. An assembly plan is constructed from an AND/OR graph, a DAG, by culling nodes and edges to yield a tree that satisfies the following conditions: each non-trivial subassembly has a single child group, which is a pair of and-siblings; the root node is the fully assembled object; and each leaf is a constituent part. Figures 2b and 2c depict two such trees. Any such tree represents a valid assembly plan, giving a partial ordering of actions to perform the complete assembly. Any sequence that obeys the partial ordering comprises a valid assembly. Since several such partial orderings are typically possible, a heuristic function provides preferences.

The planning problem is specified by an initial state and a desired goal state, where a state is a partition of the set of parts into subassemblies. The initial state may be either the completely disassembled item or some intermediate state, such as after the worker has finished an assembly operation and must replan. A traditional heuristic search algorithm such as A* may be employed to solve such an assembly problem, although optimality results in poor scalability for large assemblies. Other search algorithms like AO* and FF scale better at the cost of losing the optimality guarantee. The choice of heuristic for any of these algorithms affects the parallelizability of the resulting plan. Section IV provides details on this algorithm in the context of a distributed assembly planner.

C. *Planning Scope*

AND/OR graph-based assembly plans omit certain details in order to provide a concise, complete assembly plan. Specifically, the representation does not provide a sufficient level of detail to plan *how* parts are joined together. We chose to exclude from the graph small fastener parts – pegs and screws – because they are handled differently from large parts. Even if fasteners were included, though, the AND/OR graph does not contain information about the manipulation skills and tools needed to attach two parts together. Furthermore, it does not specify the number of robots or their roles to execute an assembly step.

In order to complement the AND/OR graph's role as a high-level planner, a lower-level planner can be used to determine the details for the execution of each step. This planner would not need to understand the complete sequence of steps in the assembly of an object, but only the details pertaining to each specific operation in the supplied sequence. This low-level planner would need symbolic planning and motion planning components.

IV. Distributed Planning Algorithm

Just as humans plan approximately in order to retain flexibility [9], in our algorithm the robots maximize flexibility by reacting to one another's actions on the granularity of a single assembly step. Alg. 1 conveys the main control loop for a robot planning as part of a team.

The parallelism of the plan originates in the structure of the AND/OR graph. By culling nodes and edges, the AND/OR graph planner yields a tree, giving a partial ordering of assembly steps. Parallelism is maximized by a balanced assembly tree. The tree structure in Fig. 2b causes each assembly operation to depend on the completion of the previous one, and it is therefore a strictly serial assembly sequence. By contrast, the assembly tree in Fig. 2c permits two operations to be performed in parallel.

In the following sections, we develop the details of heuristic search over an AND/OR graph, and then discuss synchronization issues, execution of the plan, and how the system is able to handle failure.

A. *Heuristic Search*

The planner employs a heuristic search over the space of assemblies. For moderate sized problems, the planner uses an implementation of the optimal A* search algorithm to develop an assembly plan. A* plans in the state space, so the algorithm elaborates a search tree of assembly states. Each state is augmented with a representation of the operation performed to arrive at this state, as well as a backpointer to the previous state in order to reproduce the final assembly sequence.

The search tree is expanded by adding states that are reachable from the current state by performing an assembly step (putting two subassemblies together). The planner uses the constraints specified by the AND/OR graph representation to add all valid states reachable from the current state by assembly to the search tree. States reachable by disassembling a subassembly are not currently considered. However, for assembly with humans or mixed human-robot teams, the planner would need to allow the possibility that a human commits an error that must be backtracked to complete the entire assembly.

We briefly describe A* planning parameters and how the assembly planning algorithm may scale to larger assembly sizes and complexities.

1) Cost: A*'s expansion step requires the computation of the cost to reach a state x' from the goal via a state x as $c(x') \leftarrow c(x) + c(x,x')$. The last term, the step cost, expresses the difficulty in combining k pairs of subassemblies in x into k larger subassemblies in state x'. The step cost defaults to unity, but particular combinations of parts may involve

a supplemental step cost due to the time or complexity involved in fastening them together. The cost of a state x, comprising a set of subassemblies, is computed as

$$c(x) = \max_{\forall a \subset x} c(a). \quad (1)$$

Thus, the cost structure prefers more parallel assemblies over more serial ones. Given a single pair of subassemblies being combined in a single A* expansion step, any additional pairings can be combined in parallel for free so long as their step costs are no more expensive.

2) Heuristic: The heuristic employed by A* currently seeks to maximize parallelism by estimating the minimum number of steps in the critical path.

The planning algorithm uses a path heuristic that measures the minimum number of assembly steps needed to reach the goal state from any state. The heuristic is defined as

$$h(x) = \lfloor \log_m \sigma(x) - \log_m \sigma(g) \rfloor, \quad (2)$$

where $\sigma(x)$ gives the number of subassemblies in state x. This value $\log_m \sigma(x)$ represents a lower bound on the number of assembly steps needed to achieve the goal state from state x, assuming maximal parallelization. This heuristic is admissible since by definition, it takes at least $\log_m k$ steps to assemble an object with k parts, assembled m parts at a time. We consider only pairwise assembly, so let $m = 2$. A* search algorithm with this heuristic guarantees the optimal solution to the assembly problem.

3) Scalability: The runtime of our search algorithm is the runtime of A* on a graph of nodes composed of the various possible states. The number of possible states is exponential in relation to the number of base parts (in the worst case), and the A* algorithm is, in the worst case, exponential in the shortest path from initial state, s, to desired goal state, g. For smaller assembly problems (such as the 12-part Ekorre assembly test), the time to actually assemble two subassemblies greatly dwarfs the time to plan the assembly process. For large assembly problems, it might be more desirable to use a non-optimal, but faster algorithm for planning, such as the Fast-Forward heuristic [4].

Optimal search algorithms like A* become prohibitively expensive to execute for larger problems. Several alternative search algorithms sacrifice optimality for tractability. The AO* search algorithm [1] is specifically designed for search directly on AND/OR graphs. Alternative heuristics, such as FF [7], may also be used to further speed up planning.

B. Synchronization

During the course of a distributed assembly process, each member of the team makes a series of individual choices about what assembly sequence to follow and which members perform which actions. Finding consensus requires synchronization among the workers.

The most fundamental synchronization job is to maintain consensus on the state of the assembly task. Before beginning to plan, each node first synchronizes its belief of the state of the assembly, which may employ sensors, inter-robot communication, or simulation data. This state update is reflected in the AND/OR graph structure. The robots replan before executing each action. Rather than adapting in real time to changes in state, a robot plans and executes based on a snapshot of the system state taken before planning each action. Additional synchronization is therefore necessary to cope with deferred state updates.

Algorithm 1 Algorithm for collaborative task execution.

```
procedure distributed_assembly(G – goal condition, P – parts set)
  while Not in goal state do
    Sync state of physical environment
    Sync state of actions by other nodes
    Determine effective actionable and plannable sets
    Plan.
    Register and validate planned action with other nodes.
    if lock acquired then
      Execute.
      Report completion
```

After generating a plan, the system identifies assembly steps that can be executed in parallel. The planner returns each group of parallel steps as a ready list. The synchronization step occurs after planning and before execution. Each robot selects an item from the list and attempts to lock the corresponding subassemblies in the AND/OR graphs of *all* robots. The system employs a three-phase commit process to ensure that the robot has an exclusive lock before beginning to execute the step. A robot may fail to acquire the lock on some graph node if the graph structure changed or if another robot has locked the same node. In such case, it updates the state, replans and tries again to lock. Since the synchronization process is somewhat costly, action selections are drawn randomly from the ready list to reduce the incidence of race conditions.

Some state updates can be predicted as a result of other robots' assembly actions. Such anticipated state updates can be incorporated into the planning process. Recognizing that we expect success in an assembly step to be more common than failure, the planner treats any parent node in the AND/OR graph that is locked at the onset of planning as having been successfully assembled. After receiving a plan, the robot updates its state once more. The robot then chooses randomly among only those actions in the ready list that are valid at this point.

C. Execution

The problem of autonomously executing single steps of an assembly plan is itself an immense topic, requiring further developments in manipulation skills. One point germane to his paper concerns parallelism within a single step. Many actions require close coordination among several workers, but this planner delegates each step to only a single worker. Workers in need of help may recruit it from among the available workers, thus achieving a higher effective parallelism than is revealed by the assembly plan. We observe this phenomenon with human teams below in Section V-B.

After executing an action successfully, the robot publishes a global state update to its locked nodes, indicating that it

successfully combined several subassemblies into one.

D. Handling Failure

Failures in assembly can occur for many reasons, such as problems in manipulation, parts being unavailable, and failures in perception. Many of these failures are of a transient nature, so it is generally fruitful to defer a failure rather than giving up. Several resolutions are available to the team, including trying the action again, trying alternative action, and seeking help from a human.

When execution completes, the robot then unlocks the graph nodes, whether the action was successful or not. In case of failure, the and-siblings are then available for all robots either to retry the same action or to perform two separate actions in which the nodes are or-siblings. In future work, past failures will be used to bias the plan toward alternative actions in the case of repeated failures.

V. Evaluation

We performed several pilot studies to determine the effectiveness of performing distributed assembly planning with an AND/OR graph. First, we performed a simulation study to gather statistically significant results. Second, we ran a live user study to gain insights into how human users perform under direction from the assembly planner. We ran each experiment on two simple target furniture kits from IKEA. The Sundvik children's chair has 4 large wooden parts, 6 wooden pegs, and 6 metal screws. The Ekorre rocking moose has 12 large wooden parts, 4 wooden pegs, and 17 metal screws.

A. Simulation Trials

We tested the distributed planner in simulation over large numbers of trials to achieve statistical significance. We modeled execution times as samples as IID samples from a truncated Gaussian distribution. Failures occurred with a 20% chance. We simulated team assemblies with team sizes from one to six. For each team size, we simulated 500 trials. For each trial, we then noted the total elapsed time to complete the assembly as well as the utilization fraction of the workers. Simple probability distributions indicated the elapsed time to complete each assembly task.

The curves in Figs. 6 and 7 indicate team performance on the IKEA Ekorre rocking moose on a range of team sizes. We observe that with increased worker count, the elapsed time asymptotes at approximately 40% of the time required for a single worker, reflecting the critical path of the assembly. Within the space of useful time sizes, worker utilization drops nearly linearly.

B. Human Trials

We performed a preliminary human trial to validate the simulation results. The simulation study indicated diminishing returns for the Ekorre assembly beyond three workers. We therefore studied team sizes of one and three untrained individuals. Each member of the team was presented with a GUI display like Fig. 4 on a separate laptop computer.

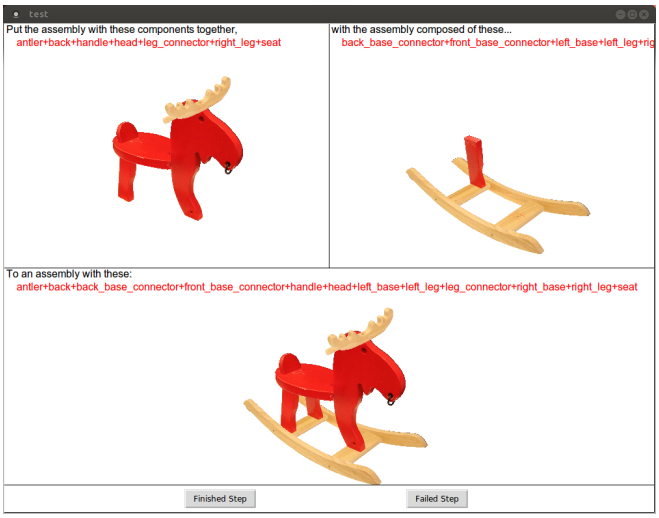


Fig. 4: GUI instructing the user to connect two subassemblies in order to complete the assembly task.

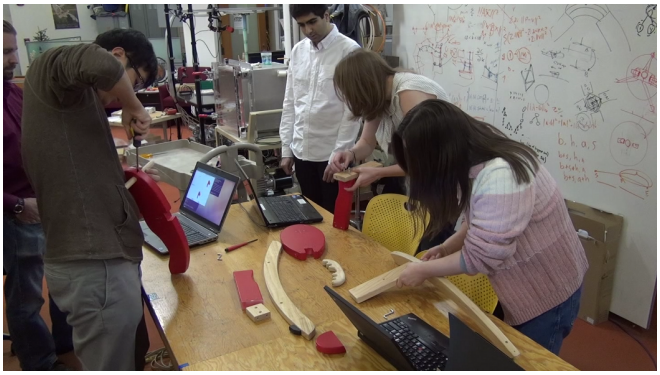


Fig. 5: Scene from our preliminary user trial.

Each computer ran the distributed assembly planner and synchronized plans before displaying the next assembly step on the screen for the user. The user clicked "Finished step" upon completion or "Failed step" if they were unable to complete it. The only failed step occurred because one piece of attachment hardware was missing from our Ekorre set. In this instance the system planned an alternative assembly sequence that did not require the step.

The results of the preliminary user trials are plotted on Figs. 6 and 7 for comparison with the predictions from the simulation study. Although the human study is insufficient to claim statistically significant results, the exercise was valuable to observe how the coordination plays out in practice.

C. Discussion

We asked each user whether they found the computer's instructions useful. Several users indicated that they appreciated the help and preferred it over figuring out the assembly sequence from the parts alone without instructions. One user stated that for the simple 12-part Ekorre kit, the instructions were unnecessary.

In fact, the system is built to cooperate with human users who have their own ideas of how to assemble furniture. Con-

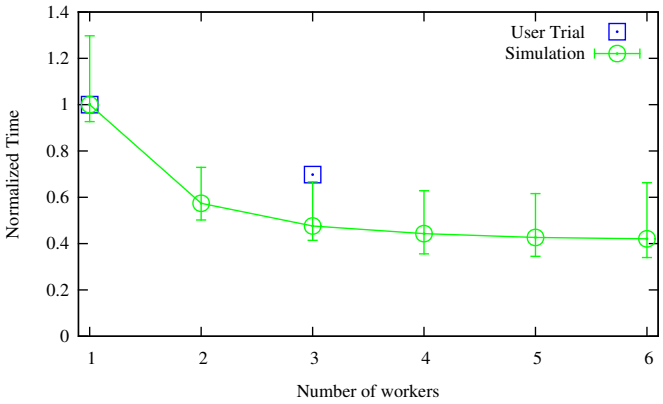


Fig. 6: Normalized comparison of assembly time for the Ekorre kit as a function of the number of workers. Simulation results are plotted against the preliminary user trial.

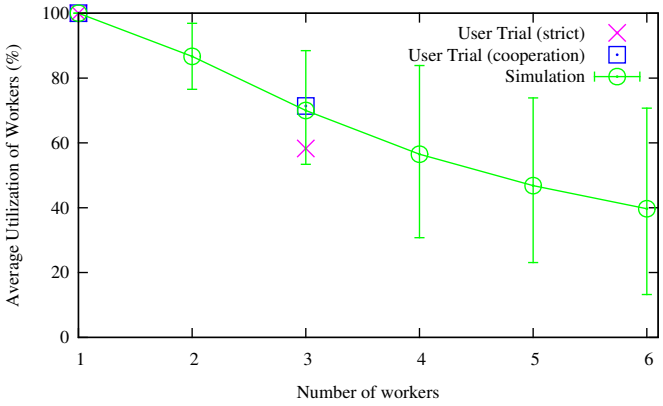


Fig. 7: Utilization of assembly time for the Ekorre kit as a function of the number of workers. Simulation results are plotted against the preliminary user trial. Here strict utilization assumes that users were idle when not assigned a task by the computer because none was available. In actuality, the users cooperated without prompting on the large, complex assembly tasks near the end, increasing the effective utilization.

sequently, the robot companions are able to plan compatible operations with mechanically-inclined users who prefer to ignore the computer's instructions and instead follow their own plan.

VI. Discussion and Future Work

The potential for parallel operation and the consequent boost in performance depends largely on the design of the item to be assembled. However, realizing that parallelism can be a challenge in a decentralized operating context involving a mix of human and robots. In this paper, we presented a system capable of recognizing and exploiting potential parallelism based on the structure of a complex object to be assembled.

An assembly planning system, when coupled with perception that can ascertain the current state of the assembly task, has a variety of applications. It can direct a team of robots to perform the assembly. It can direct a robot to cooperatively assemble items together with a human. It can act as an interactive instruction manual by guiding a human through the assembly process.

Several avenues of future work are apparent. We plan to continue development of the AND/OR graph planner with more scalable algorithms than A*. We plan to continue development of user interfaces to allow humans to coordinate actions with the robot. We are especially interested in giving directions to users who may have limited situational knowledge about the overall problem in order to ask for help following a failed assembly step. Finally, we must develop many new manipulation skills and tools to autonomously execute the assembly steps generated by the planner for a variety of objects.

References

[1] Luiz S. Homem de Mello and Arthur C. Sanderson. And/or graph representation of assembly plans. *IEEE Transactions on Robotics and Automation*, 6(2), April 1990.

[2] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[3] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246, 2006.

[4] Jörg Hoffmann. FF: The fast-forward planning system. *AI magazine*, 22(3):57, 2001.

[5] Ross A. Knepper, Todd Layton, John Romanishin, and Daniela Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 5 2013.

[6] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL-the planning domain definition language. 1998.

[7] Bernhard Nebel and Jörg Hoffmann. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[8] David Stein, T Ryan Schoen, and Daniela Rus. Constraint-aware coordinated construction of generic structures. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, San Francisco, USA, September 2011.

[9] Lucille Alice Suchman. *Plans and situated actions: the problem of human-machine communication*. Cambridge university press, 1987.

[10] Stefanie Tellex, Ross A. Knepper, Adrian Li, , Thomas M. Howard, Nicholas Roy, and Daniela Rus. Asking for help using inverse semantics. In *Proceedings of the Robotics Science and Systems Conference*, Berkeley, USA, 7 2014.

[11] Randall H Wilson and Jean-Claude Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1994.