18th IEEE International Conference on
Autonomous Robot Systems and Competitions (ICARSC)
April 25-27 2018, Torres Vedras, Portugal

Automatic Generation of Disassembly Sequences and Exploded Views from SolidWorks Symbolic Geometric Relationships

Carlos M. Costa[1], Germano Veiga[1], Armando Sousa[1], Luís Rocha[1],
Eugénio Oliveira[2], Henrique Lopes Cardoso[2] and Ulrike Thomas[3]

***Abstract*—Planning the optimal assembly and disassembly sequence plays a critical role when optimizing the production, maintenance and recycling of products. For tackling this problem, a recursive branch-and-bound algorithm was developed for finding the optimal disassembly plan. It takes into consideration the traveling distance of a robotic end effector along with a cost penalty when it needs to be changed. The precedences and part decoupling directions are automatically computed in the proposed geometric reasoning engine by analyzing the spatial relationships present in SolidWorks assemblies. For accelerating the optimization process, a best-first search algorithm was implemented for quickly finding an initial disassembly sequence solution that is used as an upper bound for pruning most of the non-optimal tree branches. For speeding up the search further, a caching technique was developed for reusing feasible disassembly operations computed on previous search steps, reducing the computational time by more than 18%. As a final stage, our SolidWorks add-in generates an exploded view animation for allowing intuitive analysis of the best solution found. For testing our approach, the disassembly of two starter motors and a single cylinder engine was performed for assessing the capabilities and time requirements of our algorithms.**

I. Introduction

Sequence planning for assembly and disassembly operations is a NP-hard problem in which the main goal is to find a physically valid sequence that respects the precedence relationships between parts while also minimizing a cost function for a specific domain problem. It has an extended set of industrial applications that follow the life cycle of any product, from the initial prototype to the production, maintenance and recycling stages. Solving this problem is especially important when we need to generate structured knowledge for an automated robotic assembly line or when we need to parallelize a set of operations in order to reduce the overall production time. Moreover, maintenance and recycling tasks can also benefit from optimal disassembly planning when it is necessary to remove a set of parts that are obstructing the retrieval of a specific group of components.

In order to tackle this problem, a SolidWorks add-in was implemented for finding the optimal disassembly sequence using a branch-and-bound algorithm implemented with a best-first search approach. Unlike most state-of-the-art in this domain, the proposed system can automatically generate the precedence relationships and disassembly directions by analyzing the geometric mating information between parts while performing a feasibility test using a collision / interference algorithm. Moreover, the inverse of the optimal disassembly sequence found can also be used for optimal assembly of products. The main reason to tackle this problem as a disassembly sequence optimization is related to the fact that every search tree branch will result in a feasible solution, whereas a considerable amount of branches of an assembly search tree will result in non-viable solutions (because the insertion of some parts may block the assembly of others). This approach allows to reduce the computational cost of the search process by reducing the number of tree nodes to evaluate while also finding the initial solution faster.

In the next section, a brief description of some related work will be given. Then, in Section III the Computer-Aided Design (CAD) analysis techniques used to extract the precedences and disassembly directions for each part will be introduced. Later on, Section IV will describe the implemented recursive branch-and-bound algorithm for computing the optimal disassembly sequence. After the introduction of the main algorithms, the analysis of the disassembly results for a starter motor and a single cylinder engine will be presented in Sections V and VI. Finally, Section VII will provide a brief set of conclusions and possible future work.

II. Related Work

Over the years, several approaches have been proposed for generating optimal and close to optimal assembly and disassembly sequences for products depending on the domain of application and its specific optimization goals.

The first stage of an assembly planner [1] is the identification of the spatial relationships and contacts between parts [2] along with the extraction of their assembly precedences [3]. This can be achieved using a generate and test methodology [4], [5] in which the relative disposition between parts is analyzed and a set of disassembly directions are generated and later on tested for verifying their physical feasibility for decoupling the components.

After knowing the precedence relationships between components, the optimal disassembly or assembly sequence of a product can be computed using the branch-and-bound algorithm [6], [7] in which the cost function may take into consideration the physical constraints between the components

[1]Authors with the Centre for Robotics in Industry and Intelligent Systems of INESC TEC and with the Faculty of Engineering of the University of Porto, Portugal (emails: {carlos.m.costa, germano.veiga, luis.f.rocha}@inesctec.pt and asousa@fe.up.pt)

[2]Authors with the Artificial Intelligence and Computer Science Laboratory at the Faculty of Engineering of the University of Porto, Portugal (emails: {eco, hlc}@fe.up.pt)

[3]Author with the Robotics and Human Machine Interaction Laboratory at the Technical University of Chemnitz, Germany (email: ulrike.thomas@etit.tu-chemnitz.de)

while also relying in other information relevant for the particular use case under analysis. For robotic assembly, this usually means a search tree that relies on information retrieved from the detailed description of the work cell, such as the position of all the parts to assemble or disassemble along with the available robots and their tools, sensors, and higher-level perception and manipulation capabilities. Moreover, it might also be necessary to use ambient fixtures in order to ensure the mechanical stability of the product during assembly or disassembly. From this information, the cost function of the branch-and-bound algorithm may use the robot travel distance, the number of possible grasps and tool changes, along with the mechanical stability, geometric accessibility, parallelism and also the number of times it was required to move the product being assembled in order for the robot to be able to access the intended regions. This information may be stored in different data structures [8], such as AND-OR graphs for modeling precedences, in 3D CAD models for specifying geometry with spatial relations and also in knowledge databases.

When an optimal solution is not needed or it takes too much time to compute, sub-optimal algorithms can be used for finding a feasible and reasonable disassembly sequence. The first approach would be to modify the branch-and-bound algorithm [9] to search only w branches at each tree expansion (a technique known as beam search) in order to limit the combinatorial explosion that may happen when generating search trees with a high branching factor. Moreover, the clustering of similar operations that are physically close would also allow to reduce the number of decision nodes, speeding up the search and allowing to identify parallelizable tasks. If a heuristic can be used to compute an upper bound from a feasible solution, then the search can be speed up further. The system presented in [10] implemented a multi-agent system for modeling a team of human-robot partners assembling products cooperatively in which the task sequences were generated with an A* algorithm that took into consideration the execution time, resource costs (energy), risk factors (danger to humans), agent experience, reliability and attention level of the agents.

For some particular applications, a feasible and reasonable solution might be enough. As such, rule based systems [11] along with evolutionary algorithms [12] or even biologically inspired approaches such as artificial bee colony [13] or ant colony optimization [14] might be employed for quickly finding a disassembly sequence solution (that can then be used as an upper bound for optimal search algorithms).

On other use cases, it might be necessary to generate the disassembly sequence to retrieve a specified set of parts [15] (useful for recycling) or might be required an integrated path planner and sequence generator [16]. Moreover, these types of systems could also be useful for generating 3D exploded views [17] of CAD models.

III. CAD Analysis

The development of complex products relies on advanced CAD systems for modeling the mechanical shape and material properties of each component while specifying their relative disposition with high accuracy. Moreover, these systems can simulate the theoretical performance and usability of the product before it is manufactured, allowing quick generation of prototypes and iterative development with continuously improved mechanical designs.

The information that is necessary to build a CAD model can also be used for other applications besides 3D modeling. One such application is the automatic extraction of the assembly precedences and generation of the optimal disassembly and assembly sequence. For implementing this functionality within one of the most used 3D CAD products, a SolidWorks[1] add-in was developed. This engineering software was chosen as our development environment because it provides an extensive set of 3D modeling tools and also has an advanced module for simulating the mechanical movement of connected parts and test their collisions in realistic conditions. Moreover, it also has a photorealistic rendering engine which is very valuable for generating the exploded view animation associated with the estimated disassembly sequence (providing an intuitive way for inspecting the planning results of our approach).

A. Symbolic Geometric Relationships

When devising the mechanical design of a product, the first stage is the 3D modeling of each individual part using a set of primitive shapes and geometric operators. Later on, these parts are used for building complex products by assembling them together using numeric and symbolic relationships, that specify their spatial arrangement along with the type of movements that they were designed for. In SolidWorks these geometric relationships are called part mates and are grouped into the standard, advanced and mechanical mates.

The standard mates are used for specifying static relationships between parts and can be numeric in nature, such as relative distance and angular displacement, or provide higher level symbolic concepts, such as coincident, concentric, tangent, parallel and perpendicular relationships.

On the other hand, advanced mates model dynamic relationships between parts, allowing to specify the range of motions that a product was designed for. These include the limit, linear coupler and path mates. Moreover, this group of mates allows the creation of higher-level relationships that combine several standard mates. Such is the case of profile centering mates along with symmetry and width mates.

Lastly, the mechanical mates group allows to model how moving parts interact with each other and transfer movement. These include the cam follower mates that are typically found between engine valves and the camshaft, along with the gear, rack / pinion and universal joint mates for modeling rotational movement transmission. For specifying range of motion there is also the hinge, slot and screw mates.

The mates introduced earlier are specific to the SolidWorks CAD design software (chosen due to its advanced modeling and simulation capabilities and also its widespread usage in the industry). But similar functionality can be found in other

[1] http://www.solidworks.com

CAD systems. However, it should be noted that this type of symbolic information is very limited or non-existent in neutral CAD formats, such as STEP or IGES, and as such, the development of either software add-ins for extracting these geometric constraints within the CAD / Computer-Aided Engineering (CAE) products or the implementation of parsers for each CAD native format is currently required.

B. Extraction of Disassembly Directions and Precedences

The symbolic relationships described in the previous section use a set of primitive mates that share a common data structure (the IMateEntity2 class of the SolidWorks Application Programming Interface (API)) and then have specific parameters for modeling each use case. This class contains information about the mate point, direction and inner and outer radius, which allows to create 5 different mate primitives (swMatePoint, swMateLine, swMatePlane, swMateCylinder and swMateCone). From these 5 primitives, the mate direction present in the planar, cylinder and cone mates can be used for computing possible disassembly directions. On the planar mate primitive, the direction vector represents the plane normal pointing to the object outside region, which directly gives information about a possible disassembly direction (it is the inverse of the mate direction). On the cylinder and cone mate primitives, the direction vector specifies the axis, which gives two possible disassembly directions (either along the axis or in reverse). If there are other mates associated with the part in question, this ambiguity can be easily solved (for example, a screw mate is usually paired with a coincident mate, which gives information about the connection of the screw head with the object being screwed, allowing to solve the ambiguity by selecting the direction of the cylinder mate that points on the opposite direction in relation to the planar mate direction).

Given that a part can be connected to several components, and as such, can contain multiple mates, the most probable valid disassembly direction can be computed by adding the mate vectors component-wise, which will result in a fused disassembly direction (that will be normalized later on). After estimating the probable disassembly direction using the mates information, it must be physically validated by moving the part along its normalized direction by a given amount and then checking if there was any collision / interference with other parts. If there was not, then the disassembly direction is considered physically valid. If the fused mate did not produce a valid disassembly operation, each mate direction can be used as a search fallback strategy. This approach allows to determine if a part can be disassembled or not, which implicitly gives information about the disassembly precedences. This is based on the fact that if the mates directions did not provide a valid disassembly direction, it is very unlikely that the part can be disassembled. It should be noted that for proper operation of this approach, the SolidWorks project that contains the parts to be disassembled must be physically valid, which means that a given volume should not be occupied by more than one part. If such requirement is not satisfied, the disassembly feasibility test will not work as expected because the collision checker of SolidWorks will detect component interferences that are not related to the movement of the parts that our system performed for validating the decoupling directions.

IV. Disassembly Sequence Planning

Finding the optimal disassembly sequence is a NP-Hard problem due to the combinatorial explosion of possible solutions that happens as the number of parts to disassemble increases. Moreover, computing the precedences between components and the parts decoupling directions requires the execution of computationally demanding algorithms (such as collision checking and mesh interference) that are increasingly slower as the volume and complexity of the geometry to test raises. In the worst case, it might be necessary to evaluate up to $n!$ search tree nodes when planning the optimal sequence for a product with n parts. Additionally, in each tree node it is necessary to analyze a set of likely disassembly directions (until a feasible one is found), which rises the number of collision checking operations to $k \times n!$ (in which k is the mean number of directions tested to remove the product components).

The goal of our SolidWorks add-in is to find a physically valid disassembly sequence that respects the precedence relationships while also minimizing a given cost function. To achieve this goal, it is necessary to reach each part and disassemble it with a specific tool (this information was provided as comments in each SolidWorks part, which can be entered manually or be automatically extracted from assembly manuals using Named-Entity Recognition (NER) algorithms [18]). The cost function models these goals using the Euclidean travel distance between the parts and also a penalty associated with each tool change (for example, screws require specific screwdrivers while loose objects need suitable grippers). After computing the optimal disassembly sequence with the associated precedences and removal directions, the assembly sequence can be computed by either reversing the order of the operations or by using the precedences to find a new assembly sequence using a different cost function.

A. Estimation of the Initial and Optimal Solution

For finding the optimal disassembly sequence for a given CAD product we implemented a recursive branch-and-bound algorithm using a best-first approach for the selection of the next branch to explore. We chose this search algorithm because it is the most memory efficient tree traversal method while also being able to quickly reach a good initial solution (allowing the branch-and-bound algorithm to be more effective at pruning branches). In Figure 1 we can see that the 3 feasible solutions found with our planning system allowed to prune many branches long before they reached their last tree level (resulting in a speed up of the search process). A breadth-first approach might be useful for parallelizing the search process, but since SolidWorks does not support concurrent calls to its collision checking / interference detection modules (because they use the state of the assembly project, namely which parts are suppressed) we are currently limited to a

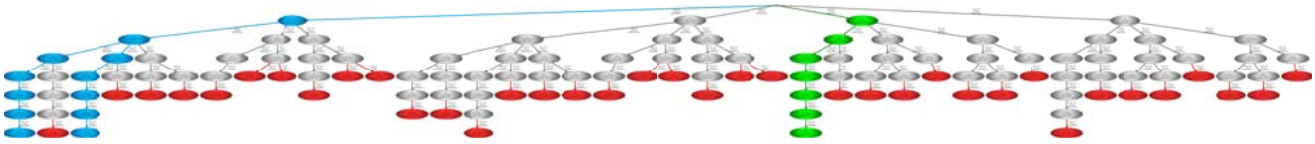

Figure 1. Small search tree for visually representing the impact that the 3 feasible solutions (blue and green branches) had on the branch-and-bound algorithm by allowing it to prune most of the tree branches (ending in red nodes) when searching for the optimal solution (green branch) for the second configuration of the starter motor

sequential approach, making the best-first search the most suitable algorithm. Moreover, given the factorial growth of the search tree, the breadth-first approach is not suitable for products with a high number of parts, because at most it needs to store all the nodes at a given tree level (while the best-first search only requires to store at most $k \times n$ tree nodes because it can be implemented as a variant of the depth-first search algorithm).

The main processing steps of our disassembly planning system are summarized in Algorithm 1. We require as input the 3D CAD models with their symbolic geometric relationships along with the tools that are needed to perform the removal of each CAD part. Our search state includes the best disassembly sequence found (with its respective cost) along with the sequence of steps that are currently applied to the SolidWorks world state (with the associated cumulative cost). The planning is started by calling the recursive procedure named $FindDisassemblyPlan$ with an empty list of cached steps. Then we check if a solution has been reached by retrieving the parts that still need to be disassembled. If none is found, then a solution has been reached and we update our search state. Otherwise, we compute which parts can be removed using the geometric reasoning engine presented in Section III in which the decoupling directions are generated by analyzing the CAD mates and validated using a move and test methodology (with collision detection for testing the disassembly feasibility). For minimizing the usage of these computational intensive algorithms we rely on a cache of steps for reusing disassembly directions that were computed previously and classified as feasible. Later on, the disassembly cost is computed for each part that can be removed and the list of steps is sorted by the disassembly cost. After knowing which parts can be removed, the one with the lowest disassembly cost is chosen and suppressed within SolidWorks followed by the recursive call to $FindDisassemblyPlan$ for achieving a best-first search tree traversal.

After reaching a solution, the recursive function call returns and backtracking is used for keep exploring the remaining of the search tree, updating the upper bound when better solutions are found until all the search tree branches are either expanded or pruned. If an optimal solution is not required, we can stop the search earlier, namely as soon as a solution is found or when a disassembly sequence has a cost lower than a given threshold. This approach is useful for products with a lot of disassembly parts which may require a long time for finding the optimal disassembly solution.

B. Speedup Techniques

Performing collision checking is a computational expensive task that should be performed only when it is strictly necessary. Given that the branch-and-bound search approach is a recursive algorithm that divides a given problem into smaller subproblems, we can reuse feasible disassembly steps on nodes with higher tree depth (counting from the start node at the top of the tree). For example, if at a given moment it was determined that it was possible to disassemble 5 components in a given set of directions, when we remove the one with the lowest cost, the disassembly directions of the other 4 will remain valid and can be cached / reused the next time we compute which components can be disassembled. However, it should be noted that this caching is only valid when expanding nodes with higher tree depth. The cached steps computed at a given search node are not valid when backtracking is performed and another tree branch is expanded. This is due to the fact that a part might be removable at a given disassembly step, but when backtracking is employed, other parts were reinserted into the collision model, invalidating the disassembly feasibility test (they may block the removal of other parts). As such, after backtracking to a parent node, the cached steps that were computed below the parent node are discarded for ensuring correct precedence and disassembly direction estimation.

C. Generation of the Exploded View

For visual inspection of the generated disassembly sequence (shown as an ordered list in the left panel of Figure 2), our disassembly system generates an exploded view within SolidWorks that animates the disassembly / assembly process (video with the animation available at[2]).

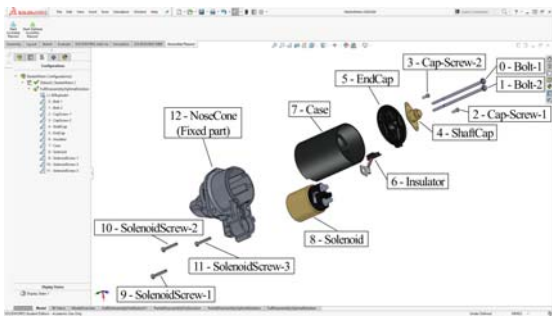

Figure 2. Exploded view generated for the starter motor with the disassembly sequence show in the numbered captions and in the left panel

[2]https://youtu.be/mK6JVkKY3z0

Algorithm 1 Disassembly Sequence Planning

System Input:
$P \leftarrow$ 3D CAD models
$M \leftarrow$ Mates associated with each CAD model
$T \leftarrow$ Tools required to disassemble each CAD model
System State:
$B \leftarrow$ Empty sequence ▷ Best solution found
$C \leftarrow$ Positive infinity ▷ Cost of best solution
$G \leftarrow$ Empty sequence ▷ Disassembly steps applied
$H \leftarrow$ Zero ▷ Disassembly cost of current solution
System procedures:
$S \leftarrow$ Cached disassembly steps
procedure FINDDISASSEMBLYPLAN(S)
  $q \leftarrow RetrievePartsToDisassemble(P, G)$
  if $IsEmpty(q)$ **then** ▷ Recursion termination
    if $H < C$ **then** ▷ New solution found
      $C \leftarrow H$ ▷ Update best solution
      $B \leftarrow G$
    return $(SolutionFound)$
  $u \leftarrow ComputeDisassemblySteps(S, q)$
  $z \leftarrow NewSet(S + u)$ ▷ Steps cache must not propagate up the function call stack when backtracking
  for all n steps in u **do**
    $f \leftarrow H + GetCost(n)$
    if $f < C$ **then** ▷ Prune branches
      $H \leftarrow f$
      $AddElement(G, n)$
      $e \leftarrow FindDisassemblyPlan(z)$ ▷ Recurse
      $H \leftarrow H - GetCost(n)$ ▷ Backtracking
      $RemoveElement(G, n)$
  return (e)
$S \leftarrow$ Cached disassembly steps
$q \leftarrow$ Available parts to disassemble
procedure COMPUTEDISASSEMBLYSTEPS(S, q)
  $k \leftarrow$ Empty set ▷ Feasible disassembly steps
  for all p parts in q **do**
    if $Contains(S, p)$ **then** ▷ Check for cached step
      $j \leftarrow ComputeStepCost(p)$
      $AddElement(k, NewStep(p, j))$
    else
      $w \leftarrow DisassemblyDirections(P, M, G, p)$
      for all d directions in w **do**
        if $PartCanMove(P, G, p, d)$ **then**
          $j \leftarrow ComputeStepCost(p)$
          $AddElement(k, NewStep(p, j))$
          break
  return $(SortByStepCost(k))$ ▷ Sort for best-first
$p \leftarrow$ Part to disassemble
procedure COMPUTESTEPCOST(p)
  $r \leftarrow DistanceToLastDisassembledPart(G, p)$
  if $ToolChangeIsRequired(T, G, p)$ **then**
    $r \leftarrow r + ToolChangeCostPenalty$
  return (r)

This functionality is useful when a mechanical engineer needs to visually analyze the assembly / disassembly sequence or needs to show it to the operators that will build the final product.

D. Generation of the DOT Graph

For allowing the detailed inspection of the nodes that the branch-and-bound algorithm has expanded, our SolidWorks add-in saves the search tree into a system file in the DOT format[3]. This file can be used to create a graphical representation of the search tree using GraphViz (example in Figure 1) or be the input for an external system that needs the generated search tree and disassembly sequence in a standard graph format.

V. DISASSEMBLY OF A STARTER MOTOR

A starter motor is a mechanical device that is used for starting internal combustion engines by rotating their crankshaft. It was chosen as a case study because it has a significant amount of precedence relationships and has a wide range of small parts that require different tools for assembling them (such as hex, Torx and Phillips screwdrivers along with a universal gripper). However, for computing optimal solutions in reasonable time, 2 different configurations containing a subset of the starter motor parts were created (and are publicaly available on our dataset[4]). In the first starter motor configuration (shown in Figures 2 to 4), the internal components were removed (namely the armature, brushes and pinion), reducing the total amount of parts to 13. For testing the impact of the initial search tree branching factor, a second configuration was created (presented in Figures 5 and 6) in which 5 more parts were removed, namely the insulator and the solenoid along with its 3 screws.

A. First and Optimal Solutions

The first solution for the disassembly of the starter motor was found using a best-first approach (introduced in Section IV-A). By expanding the best branch on each tree level during the depth-first search, we can compute an initial solution in a very short time. In Figure 3 the first solution found for the first configuration of the starter motor is presented (the green numbers / lines represent the parts original positions and the blue numbers / lines correspond to the parts pose after applying the exploded view animation that was generated by our SolidWorks add-in). The search started with a universal gripper at coordinates (0,0,0), which resulted in the best-first search algorithm selecting the bottom parts first (namely the 3 screws holding the solenoid), and then due to the disassembly precedences, it moved on to the cap screws, followed by the shaft cap, bolts, end cap, insulator, case, solenoid and finally the nose cone (this part was excluded from the planning algorithm because it was marked as fixed). In this initial solution there were 5 tool changes (with a cost penalty of 0.5 for each one), which along with the travel distance of

[3]https://www.graphviz.org/doc/info/lang.html
[4]https://github.com/carlosmccosta/AssemblyPlannerDataset

0.662 m resulted in a total disassembly cost of 3.162 when disassembling the 13 parts.

Given that the best-first search approach selects the next best candidate by analyzing the nodes at the next tree level, it may choose sequences in which the disassembly tool is exchanged more times than it needed to. As such, after finding the first solution, we backtrack on the search tree and continue exploring the other branches (pruning when possible), in order to ensure that the optimal solution is found. Looking at the optimal disassembly sequence shown in Figure 4, it can be seen that it managed to reduce the travel distance by 0.102 m while also performing one less tool change, which allowed to reduce the disassembly cost by 0.602. It should be noted that the feasibility of a disassembly operation uses the geometric mate directions along with a collision / interference detection algorithm (it does not use CAD meta information, such as identifying a component as being a screw). As such, if a screw has no thread, it will be physically possible to disassemble the parts attached to it without removing the screw first. To illustrate this behavior, the solenoid screws were modeled without thread and by analyzing the optimal sequence we can see that the solenoid was disassembled before its screws because it had a shorter travel distance in relation to the previously disassembled parts (since the disassembly was progressing in a top-down direction).

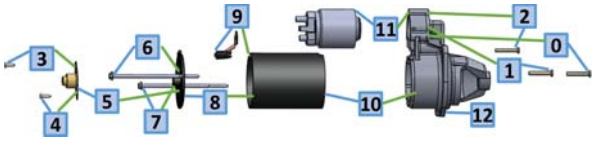

Figure 3. Disassembly sequence for the first solution found with a cost of 3.162 and travel distance of 0.662 m when dismantling a product with 13 parts (12 nodes were visited in 3 minutes and 2 seconds)

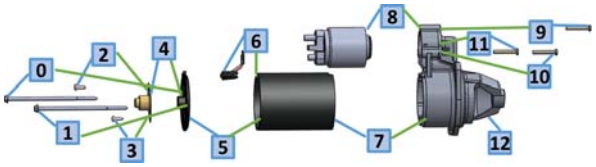

Figure 4. Disassembly sequence for the optimal solution found with a cost of 2.560 and travel distance of 0.560 m when dismantling a product with 13 parts (16339 tree expansions were performed in 30 hours, 24 minutes and 3 seconds in which 9060 branches were pruned)

Analyzing the logs and search trees generated by our SolidWorks 2016 $C\#$ add-in (compiled using Visual Studio 2017 with the 4.6.2 .NET framework) and running on a Clevo N170RD laptop (with an Intel Core i7-6700HQ CPU and Windows 10 x64), we could see that the first solution took 3 minutes to expand 12 tree nodes while the optimal solution required 36 hours, 56 minutes and 59 seconds for expanding 16339 tree nodes. During the optimal search, 5 solutions were found, allowing to prune 9060 branches (more than 50% of branches were pruned long before reaching the last tree levels, resulting in the expansion of much less than 50% of the search tree nodes).

For improving the search speed even further, we activated the caching approach introduced in Section IV-B and managed to reduced the search time down to 30 hours, 24 minutes and 3 seconds (18% time reduction) while achieving the same results. This technique can provide a significant speedup when there is a high tree branching factor and the trees have a high number of levels (each level corresponds to the disassembly of a single part). For example, in this use case, we can see that in the beginning we can disassemble 7 parts (2 cap screws, 3 solenoid screws along with 2 long bolts). When we remove one of these parts, the disassembly directions of the other 6 remain valid (they are a subproblem), and can be reused if we continue the disassembly process (no need to compute the disassembly directions and perform collision checking again), which can lead to considerable speed ups given that the most computationally expensive task is the collision checking / interference detection stage.

B. Impact of Part Count and Branching Factor

In order to analyze the impact that the number of disassembly parts and the search tree branching factor have on the computation time required to find the optimal solution (and also be able to generate a search tree small enough to be readable in a figure), a second configuration of the starter motor was created in which 5 parts were removed (in relation to the first configuration). Three of these removed parts could be disassembled on the first search tree level (namely the 3 screws holding the solenoid), and the other two were on the middle and end of the search tree (which were the insulator and the solenoid respectively).

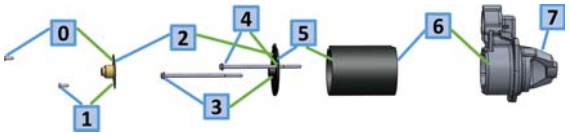

Figure 5. Disassembly sequence for the first solution found with a cost of 2.290 and travel distance of 0.290 m when dismantling a product with 8 parts (7 nodes were visited in 1 minute)

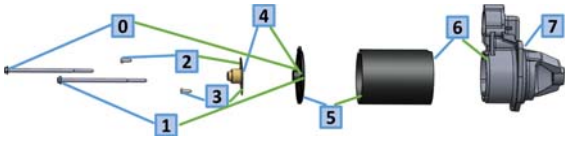

Figure 6. Disassembly sequence for the optimal solution found with a cost of 1.771 and travel distance of 0.271 m when dismantling a product with 8 parts (122 tree expansions were performed in 11 minutes and 27 seconds in which 37 branches were pruned)

Analyzing Figure 5 we can see that the first solution was a top-down disassembly sequence, starting with the cap screws, then removing the shaft cap, bolts, and finally extracting the loose objects below (namely the end cap, followed by the case and nose cone). On the other hand, the best solution (presented

in Figure 6), managed to disassemble the 8 parts with one less tool change while also reducing the travel distance by 0.019 m (it extracted all the screws first and then removed all the parts with a universal gripper).

From a time performance point of view, reducing the tree depth and initial branch factor had a tremendous effect on the number of visited nodes, due to the factorial tree growth and also because the initial tree levels will not be pruned (because they have a cost lower than the optimal solution). In this particular use case, reducing from a total of 13 to 8 parts (from which the number of parts that could be disassembled in the first tree level was reduced from 7 to 4), we can see in Figures 3 to 6 that the time needed to find the first solution was reduced by 67% (from 3 minutes and 2 seconds to 1 minute) while the time required to compute the optimal solution was reduced by 99.37% (from 30 hours 24 minutes and 3 seconds to 11 minutes and 27 seconds). These results show the combinatorial explosion that disassembly planning systems face, and as such, for trying to find the optimal solution for very large products it is recommended to break the search problem into a set of small optimization problems, such as planning the disassembly sequence for subcomponents and then running the global optimization on top of large clusters of parts. This approach would allow to exploit the spatial locality of disassembly operations and bound its combinatorial explosion for products with a large number of components.

VI. Disassembly of a Single Cylinder Engine

For assessing the suitability of our best-first search algorithm for disassembling products with a large set of parts we added to our set of benchmarks the analysis of the disassembly sequence generated for dismantling a single cylinder engine with 40 components (solution found in 1 hour, 57 minutes and 35 seconds and shown in Figures 7 to 9). Inspecting Figure 7 we can see that the tool change cost of 0.5 along with the compact design of the single cylinder engine (bounding box dimensions with less than 0.5 meters) caused the best-first search algorithm to group components by the type of robotic end effector that was needed to remove them (because the tool change cost penalty was higher than the maximum distance between parts). This resulted in the removal of all of the 22 large bolts (shown in Figure 7) using a 8 mm spanner tool followed by the extraction of the outer engine components (illustrated in Figures 8 and 9) with an universal gripper. Then, the search algorithm continued to remove the parts seen in Figure 9 until no more components could be removed with the universal gripper. Later on, a 6 mm spanner tool was used to extract the two bolts holding the chain tensioner followed by its removal with an universal gripper. Given that our SolidWorks add-in temporarily disables all part mates in order to be able to move the components during the disassembly feasibility test, it currently assumes that each disassembly operation results in the extraction of a single part unit (with nothing else attached to it). As such, the large components seen in Figure 9 could only be removed after the two bolts with large diameter were extracted (using a 4 mm Allen wrench). In the future we intend to devise a more advanced algorithm to test the disassembly feasibility for allowing the removal of components that still have other parts attached to them. This functionality could be implemented by selectively disabling part mates while moving the components in each decoupling direction (in order to bring attached parts along). This extension to our SolidWorks add-in could prove very useful for parallelizing tasks and could also be included in the cost function of the assembly planner for prioritizing sequential or parallel disassembly sequences.

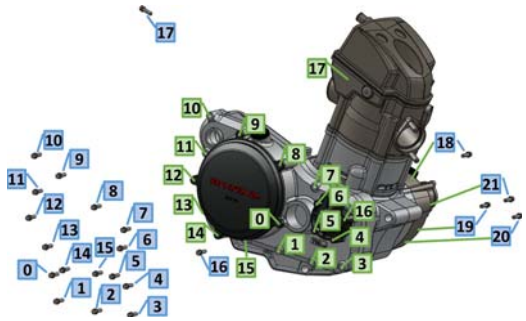

Figure 7. Exploded view for the first set of disassembly operations found by the best-first search algorithm (consisting of the removal of all the bolts with a head of 8 mm)

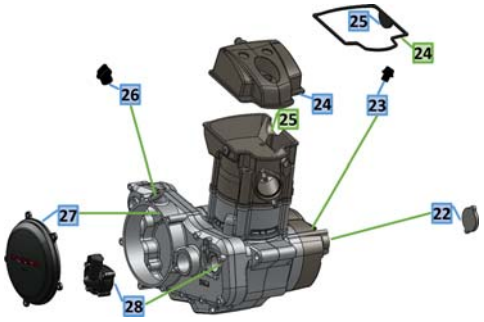

Figure 8. Exploded view for the second set of disassembly operations found by the best-first search algorithm (consisting of the removal of the outer parts)

Figure 9. Exploded views for the third (left) and fourth (right) set of disassembly operations found by the best-first search algorithm (consisting of the removal of the large center blocks)

Performing an overall analysis of Figures 7 to 9, we can observe that the usage of a high tool change cost along with

the travel distance of the robotic end effector made the best-first search algorithm behave similarly to a human operator, since it tries to minimize the number of times it changes the disassembly tools while also minimizing the effort / distance between each consecutive operation. As such, this approach may prove fruitful for teaching or assisting human operators using augmented reality interfaces and could also provide a useful plan for a robotic assembly line (which can later on be improved by the branch-and-bound algorithm and be coupled with arm motion path planning and grasping systems).

VII. Conclusions

The generation of the optimal sequence for disassembling or assembling products is an NP-hard problem with a wide range of applications in the manufacturing industry (such as the automation of assembly lines using robotic arm manipulators). This paper presents a set of techniques for speeding up the search of the optimal solution using caching / reuse of disassembly steps along with a best-first search approach for estimating the initial solution for the branch-and-bound algorithm. Moreover, by analyzing the geometric mates found on SolidWorks CAD assemblies, along with collision / interference detection algorithms, our SolidWorks add-in can automatically compute the part decoupling directions and the disassembly precedences for generating physically feasible disassembly sequences. For intuitive use of the system, it also creates an exploded view animation associated with the best solution found in order to allow the inspection of the disassembly and assembly sequence.

Depending on the search tree branching factor and the number of components to disassemble, the estimation of the optimal solution may take from a few minutes to a few hours. For very complex products with a high number of parts, a heuristic method may provide reasonable results with a much shorter computation time. As such, future work could include the development of a heuristic algorithm to find close-to-optimal disassembly sequences and also the exploration of clustering techniques for exploiting the spatial locality of large disassembly operations and bound their search tree combinatorial explosion. Moreover, it would be interesting to extend the branch-and-bound cost function to model other disassembly factors, such as possible grasp positions for the parts, simulation of gravity and how it affects the mechanical stability of the components along with the usage of fixtures (among many others). On a higher level, it would also be useful to integrate an arm motion path planner for allowing the generation of the structured knowledge required for a robotic manipulator to perform the disassembly / assembly operations.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 723658.

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project SAICTPAC/0034/2015-POCI-01-0145-FEDER-016418.

References

[1] D. Whitney, *Mechanical assemblies: their design, manufacture, and role in product development*. Oxford University Press, 2004.

[2] P. Tang and J. Xiao, "Automatic generation of high-level contact state space between 3d curved objects," *The International Journal of Robotics Research*, vol. 27, no. 7, pp. 832–854, 2008.

[3] G. Pintzos, C. Triantafyllou, N. Papakostas, D. Mourtzis, and G. Chryssolouris, "Assembly precedence diagram generation through assembly tiers determination," *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 10, pp. 1045–1057, 2016.

[4] S. G. Kaufman, R. H. Wilson, R. E. Jones, T. L. Calton, and A. L. Ames, "The archimedes 2 mechanical assembly planning system," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, 1996, pp. 3361–3368.

[5] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3, pp. 577–601, Apr 2000.

[6] U. Thomas and F. Wahl, "Assembly planning and task planning - two prerequisites for automated robot programming," in *Robotic Systems for Handling and Assembly*. Springer, 2011, pp. 333–354.

[7] U. Thomas, T. Stouraitis, and M. A. Roa, "Flexible assembly through integrated assembly sequence planning and grasp planning," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Aug 2015, pp. 586–592.

[8] U. Thomas and F. M. Wahl, "A system for automatic planning, evaluation and execution of assembly sequences for industrial robots," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2001, pp. 1458–1464.

[9] X. F. Zhang and S. Y. Zhang, "Product cooperative disassembly sequence planning based on branch-and-bound algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 51, no. 9, pp. 1139–1147, 2010.

[10] L. Johannsmeier and S. Haddadin, "A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 41–48, Jan 2017.

[11] S. Smith and W. Chen, "Rule-based recursive selective disassembly sequence planning for green design," *Advanced Engineering Informatics*, vol. 25, no. 1, pp. 77–87, Jan. 2011.

[12] A. Elsayed, E. Kongar, S. M. Gupta, and T. Sobh, "A robotic-driven disassembly sequence generator for end-of-life electronic products," *Journal of Intelligent and Robotic Systems*, vol. 68, no. 1, pp. 43–52, Sept. 2012.

[13] W. Yuan, L. Chang, M. Zhu, and T. Gu, "Assembly sequence planning based on hybrid artificial bee colony algorithm," in *IFIP Advances in Information and Communication Technology*. Springer, 2016, pp. 59–71.

[14] X. Liu, G. Peng, X. Liu, and Y. Hou, "Disassembly sequence planning approach for product virtual maintenance based on improved max–min ant system," *The International Journal of Advanced Manufacturing Technology*, vol. 59, no. 5, pp. 829–839, Mar 2012.

[15] S. Smith, G. Smith, and W. Chen, "Disassembly sequence structure graphs: An optimal approach for multiple-target selective disassembly sequence planning," *Advanced Engineering Informatics*, vol. 26, no. 2, pp. 306 – 316, 2012.

[16] D. T. Le, J. Cortes, and T. Simeon, "A path planning approach to (dis)assembly sequencing," in *2009 IEEE International Conference on Automation Science and Engineering*, Aug 2009, pp. 286–291.

[17] W. Li, M. Agrawala, B. Curless, and D. Salesin, "Automated generation of interactive 3d exploded view diagrams," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 101:1–101:7, Aug. 2008.

[18] C. M. Costa, G. Veiga, A. Sousa, and S. Nunes, "Evaluation of stanford ner for extraction of assembly information from instruction manuals," in *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, April 2017, pp. 302–309.