To appear in the *International Journal of Production Research*
Vol. 00, No. 00, August 2014, 1–18

Article

An improvement heuristic framework for the laser cutting tool path problem

Reginald Dewil[a]*, Pieter Vansteenwegen[a], Dirk Cattrysse[a], Manuel Laguna[b] and Thomas Vossen[b]

[a]*KU Leuven, Centre for Industrial Management/Traffic & Infrastructure, Celestijnenlaan 300a, 3001 Leuven, Belgium*; [b]*CU Boulder, Leeds Business School, USA*

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

This paper deals with generating cutting paths for laser cutting machines by representing a tool path in a novel way. Using the new representation, the tool path problem can be viewed as finding a partitioning of contours which minimizes the sum of the costs of a rooted directed minimum spanning tree to connect the partitions and the costs of a generalized traveling salesman problem (GTSP) solutions within each partition. Using Edmond-Liu's algorithm to solve the arborescence problem, an improved Lin-Kernighan heuristic to solve the GTSP and a heuristic repartitioning approach, tool paths can be generated that are 4.2% faster than those generated by an existing tool path construction heuristic.

Keywords: laser cutting ; tool path ; PCGTSP ; RDMST

1. Introduction

Many manufacturing processes start with cutting parts from a stock sheet with laser cutting being one of the more flexible cutting methods. Automated CAM software can create efficient part nests to minimize waste material and provides tool paths to guide the laser head in cutting all the nested parts. The generation of tool paths for laser cutting machines is the focus of this paper as optimizing the tool path quality may lead to substantial savings in terms of money and resources. A typical cutting process can take anywhere from several minutes to several hours, depending on the number of parts on the plate, the material type, the process parameters, and the plate thickness. The repositioning of the cutter head between cuts, called the airtime motion, depends on the chosen tool path and is to be minimized as it is non-productive time.

Hoeft and Palekar (1997) classify tool path problems into three problem classes:

- ***Continuous Cutting Problem*** *(CCP)*: the cutter head visits each contour to be cut once. The tool can engage the contour at any point on its perimeter, but must cut the entire contour before it travels to the next contour. Accordingly, the same point must be used for entry and departure from the contour.
- ***Endpoint Cutting Problem*** *(ECP)*: the tool can enter and exit contours only at some predefined points on the boundary. However, it may cut the contour in sections, or stated otherwise: a contour can be pre-empted.
- ***Intermittent Cutting Problem*** *(ICP)*: this is the most general version of the problem in which contours can be pre-empted and there is no restriction on the points that can be used for entry or exit.

These three problem classes cover a wide range of tool path problems. However, the following important case is not considered: the tool path visits each contour to be cut once and the tool can

*Corresponding author. Email: reginald.dewil@kuleuven.be

engage the contour only at some predefined points on the boundary. This problem corresponds to the ***Generalized Traveling Salesman Problem*** *(GTSP)* introduced by Srivastava et al. (1969). The GTSP consists of a set of cities grouped in districts and the objective is to find the shortest route visiting exactly one city of each district. The four problem classes are shown in figure 1.

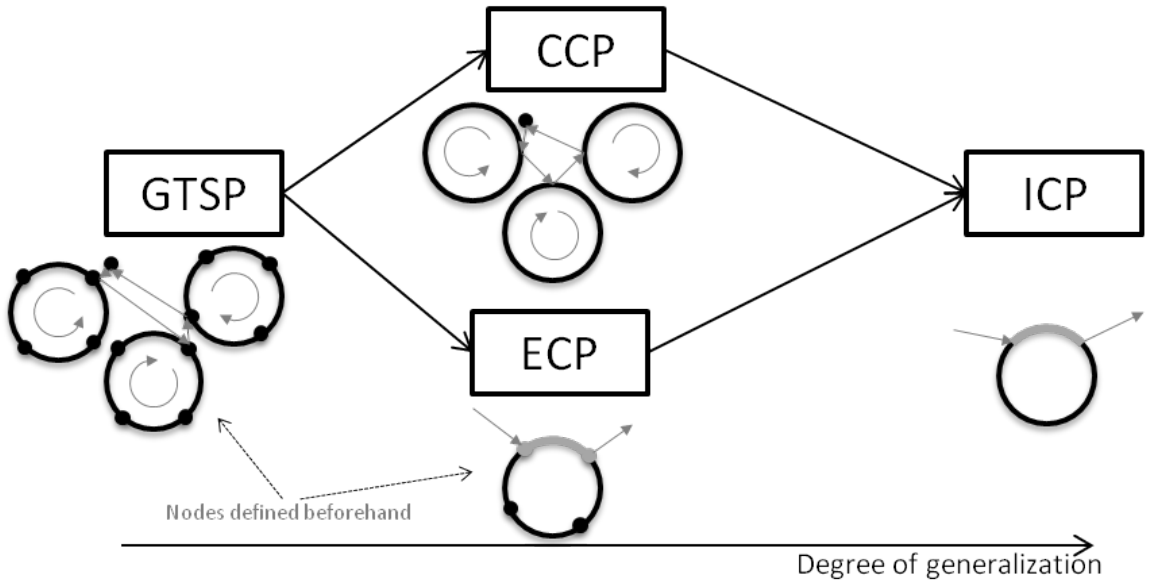

Figure 1. Different versions of the cutting path problem

Previous research on tool path problems has mainly dealt with either the GTSP or the CCP versions of the cutting path problem. This is most likely the result from the fact that many cutting processes don't allow pre-emptions because of quality reasons i.e. when a contour is pre-empted it is hard for some cutting processes to restart the cutting process later on at exactly the location where the cutting tool exited the contour. In laser cutting, however, this is possible. Moreover, as will be explained in the next section, because of the existence of *common cuts*, contour pre-emption is unavoidable.

The GTSP version is considered by Raggenbass and Reissner (1989, 1991), Han and Na (1999a,b), Xie et al. (2001), Castelino et al. (2003), Kim et al. (2004), Wang and Xie (2005), Vaupotic et al. (2006), Xie et al. (2009), Yang et al. (2010), and Jing and Zhige (2013).

The CCP version is considered by Hoeft and Palekar (1997), Veeramani and Kumar (1998), and Lee and Kwon (2006). The above approaches iteratively switch between a part sequence determination (PSD) problem and a pierce point determination (PPD) problem. The major differences between the above approaches lie in the selection of the heuristics tackling each sub problem.

In the cutting path problem, parts can have holes in them. Efficient nesting algorithms can in turn nest other parts in these holes in order to minimize waste material. In general, one can state that contours can have inner contours. These inner contours need to be cut before the outer is completely cut because when a contour is cut completely, it detaches from the rest of the plate. If there is a supporting grid, it can possible shift its position, or if there is no supporting grid, it simply falls through. In both cases, it will be impossible to continue cutting in this area. In the above approaches, this precedence constraint is handled by forcing inner contours to be cut immediately before its outer contour and as such it does not introduce much extra complexity.

Both the ECP and ICP have received far less attention in the literature than the GTSP or CCP. With the exception of a problem description of the Crossing Postman Problem by Garfinkel and Webb (1999), which can be viewed as a kind of ICP, no previous solution approaches for the ICP have been identified.

A special version of the ECP has been considered for wire electric discharge machining (EDM) by

Moreira et al. (2007) and Imahori et al. (2008). In wire EDM, the cutting tool never stops cutting, and as such, a different set of precedence constraints applies for wire EDM than for laser cutting.

Dewil et al. (2011, 2014) showed that the endpoint cutting problem can be modeled as a precedence constrained GTSP (PCGTSP). The precedence constrained version of the GTSP contains precedence constraints between cities, between districts and between cities and districts. Dewil et al. (2011) showed that high quality solutions for the laser cutting tool path problem for thin plates can be reached. These solutions are obtained through a modified insertion construction heuristic followed by a tabu search meta heuristic (Glover and Laguna (1999)) using generic 3-opt moves modified to take the relevant precedence constraints into account(Gambardella and Dorigo (2000)). However, in thicker plates additional costs and precedence constraints have to be considered such as piercings, pre-cuts and sharp angle macros. Dewil et al. (2012, 2014) showed that the generic heuristics of Dewil et al. (2011) become prohibitively computationally expensive as these additional costs and constraints are included. Dewil et al. (2014) present a set of construction heuristics that takes advantage of the problem structure in order to generate tool paths that require a considerably shorter process time than those currently generated by CAM software packages in similar computation times. In a similar approach, this paper presents a heuristic framework that utilizes the problem structure in order to improve existing tool paths.

The remainder of this paper is organized as follows. In the next section, the laser cutting problem is introduced in more detail. Section 3 shows how a tool path can be represented as a tree with cycles and section 4 presents a hybrid heuristic that takes advantage of this structure. Section 5 discusses our experimental results, while section 6 presents our conclusions.

2. Laser Cutting Tool Path Problem

The objective of the tool path problem, as defined by Dewil et al. (2014), is to minimize the total time required to cut a number of nested parts from a metal sheet. A part consists of an outer contour and possibly a set of inner contours. Each contour itself is composed of a number of elements: lines or arcs which can be cut in both directions. Additionally, a *pierce group* is defined as a single contour or a set of contours that are connected to one another through common cuts. Important characteristics of our problem setup are the following:

- The actual cutting time is considered to be independent of the chosen tool path and as such will not be considered further in this paper.
- The airtime is the sum of all non cutting movements of the laser head.
- Every time the laser head moves to a new section of the plate, the laser first needs to pierce through the plate before the laser head can start cutting. This is called a piercing and the time required to execute the piercing is called the piercing cost.
- In order to allow contour pre-emptions, so-called pre-cuts are used. A pre-cut is a short cut that is made in an element when cutting an adjacent element. It allows the laser head to start cutting the element later on from within the pre-cut without requiring a piercing.
- In thicker plates an extra cost is added when cutting a sharp angle. This cost equals the extra time required to execute a special *sharp angle macro* which is use to avoid corner burn-off or quality deterioration due to excessive pre-heating.

The total time required to cut a set of nested parts from a metal sheet consists of the time required to execute all cut moves, air moves, piercings, pre-cuts and sharp angle macros. The laser cutting tool path problem is further complicated by the presence of precedence constraints, explained in detail in Dewil et al. (2014).

As mentioned in the introduction, once a contour is completely cut, it detaches from the rest of the plate. This detached area can possibly shift position which then becomes inaccessible for further cuts. In this research, it is not necessary that an inner contour is cut immediately before its outer contour is cut. In addition, since in the laser cutting ECP a part can be pre-empted and

a contour is not cut until all of its elements are cut; it is allowed to cut some elements of a contour before cutting the inner contours. In other words, the first set of precedence constraints requires that a contour needs to be cut before the last element of its outer contour is cut.

The inner-outer contour relations, as depicted in figure 2 can come from holes in parts (a), parts nested in holes (b), or parts nested in islands (c). Islands are areas that have become completely enclosed through efficient nesting algorithms. Such algorithms position parts so close to one another that they share a cutting line which are called *common cuts*. Each common cut is, in turn, enclosed by a contour composed of the two contours it is part of. If this composite contour is cut, it can also shift and cause the common cut enclosed within to become inaccessible to cut. Therefore, a common cut needs to be cut before its composite contour is cut.

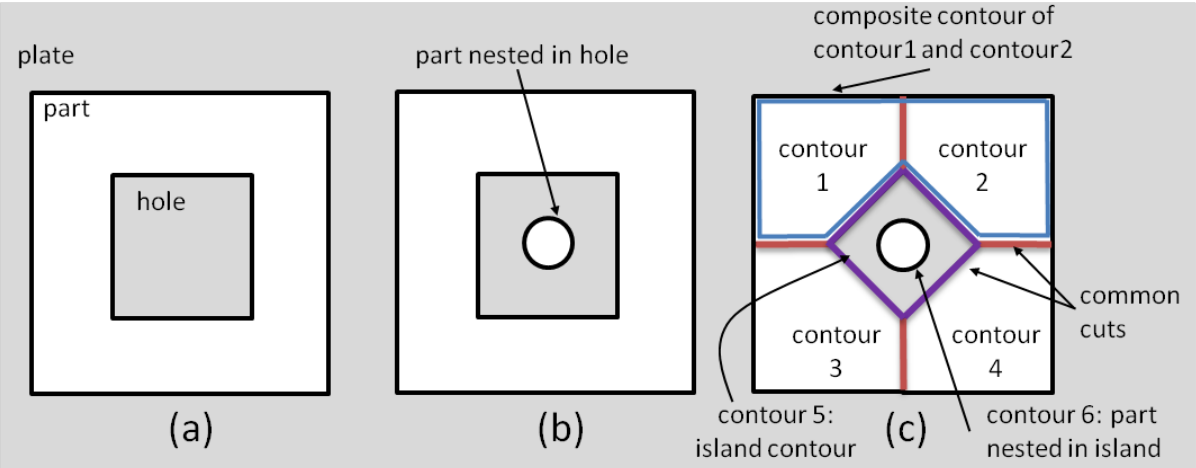

Figure 2. Parts, holes and common cuts that lead to precedence constraints (Dewil et al. (2014))

Common cuts furthermore require extra pre-cuts because when a contour is completely cut, it can shift into the cut kerf. As a consequence, adjacent elements cannot be cut up to the cut contour since this might damage the shifted contour. Dewil et al. (2014) explain this constraint in detail and propose two heuristics to generate sub tool paths for pierce groups containing common cuts. One of these heuristics (the element insertion heuristic) will be used in this research to generate the sub tool paths for common cut pierce groups.

Dewil et al. (2011, 2014) modeled the tool path as a GTSP with special precedence constraints. Each element is represented as a district of two cities where each city represents a cutting direction of the element. Generic local improvement heuristics for the GTSP were adapted to include the special precedence constraints and embedded in a tabu search framework. Because elements are considered individually, much of the optimization process is spent examining solutions that do not resemble an optimal tool path. In an optimal tool path, when the laser head enters a contour for the first time at a certain location, the laser head will eventually exit the contour from that same location, regardless if pre-emption occurs or not. As such, the first entry actually fixes the relative order and directions of elements[1] in that contour. This information is not considered in the above mentioned precedence constrained GTSP approach.

In Dewil et al. (2014), several construction heuristics were presented that take advantage of this observation, with the PFr heuristic being the best on average. The PFr heuristic starts with a list of all outermost pierce groups. It then iteratively selects the pierce group that is closest to the current partial path i.e. an insertion location, an entry point and an exit point are selected. Given these entry and exit points, a sub tool path for the pierce group is determined which is

[1]In fact two directions are possible, but from an optimization point of view, the direction actually does not matter with regard to minimizing total time. From a technical point of view, there is a benefit in preferring one direction over the other because a laser head is not perfectly symmetrical, resulting in better quality cuts on a certain side of the laser head.

trivial for a normal contour, but is more complicated for a pierce group containing common cuts. The sub tool path is inserted in the partial path at the chosen insertion location and the process continues until all pierce groups of this top level are inserted. Following this step, the heuristic then iterates backwards over the partial path and if it encounters an element that closes a contour, all of the contour's inner contours (if any) are inserted analogously to the insertion of the top level contours. When the iterator reaches the start of the path, a complete tool path has been constructed, represented by an array of cuts.

In the following section, a different tool path representation is proposed that can be exploited efficiently by improvement heuristics.

3. Tool Path Representation

Without loss of generality and for the ease of understanding, only pierce groups consisting of a single contour are considered in the following description.

Consider the tool path in 3(a), which is devoid of any pre-cuts, or put otherwise, not a single contour has been pre-empted. This is basically a GTSP solution.

By contrast, the tool path in 3(b) pre-empts all but one contour. The tool path consists of an air move *(1)* to the first contour where section *(2)* is cut, followed by air move *(3)*, section *(4)* is cut and so on. Such a tool path can be seen as an arborescence (Tarjan (1977)) if every contour would be represented by a single vertex.

Finding a least cost arborescence is also known as a rooted directed minimum spanning tree problem (MST - Kruskal (1956)). Whereas the MST consists of finding a spanning tree of minimum weight between the vertices in a connected undirected graph, the arborescence problem consists of finding a subgraph of minimum weight between the vertices in a connected, directed graph. The root vertex is the start location of the laser head on the laser cutting machine and a vertex (contour) has children if it is pre-empted to cut other contours.

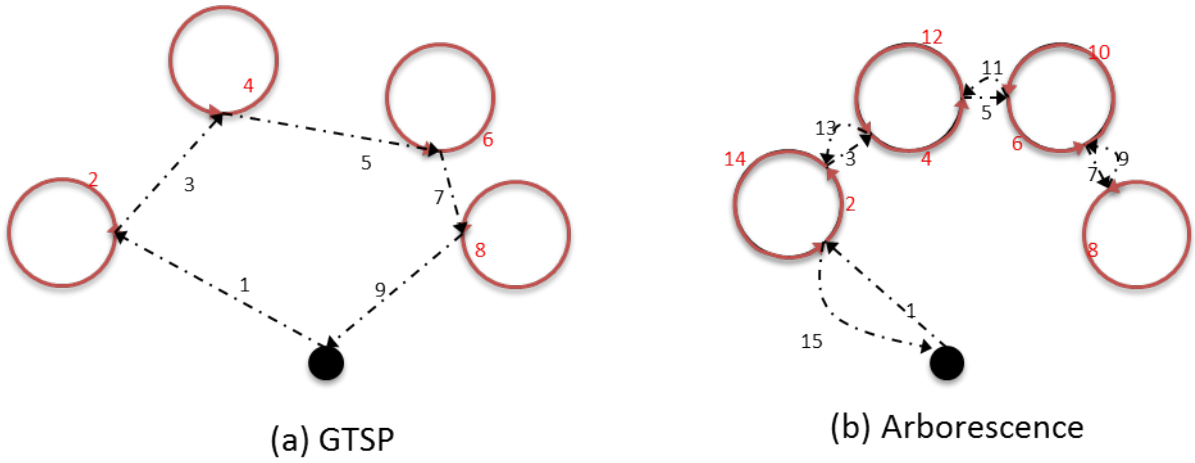

Figure 3. Two extreme types of tool paths

Most optimal tool paths do not correspond to an arborescence between pierce groups. For instance, consider the case in figure 3(a) where inter contour distances[2] (air moves) are large compared to the contour sizes[3]. In this case, if an arborescence would be constructed between the

[2]The inter contour distance between two contours i and j is the minimum of all distances between all nodes of contour i and all nodes of contour j.

[3]The contour size is an approximation to the additional length the laser head has to travel going from the preceding contour *i-1* to the succeeding contour *i+1* since in doing so, the laser head has to "cross" (at least partly) the contour i.

pierce groups, the double execution of air movements (once going to a contour and once returning from this contour) would result in high air movement costs coming from the many air moves on top of the many pre-cut costs.

Similarly, most optimal tool paths also won't be GTSP solutions. For instance, consider the case in figure 3(b) where inter contour distances are small compared to the contour sizes. In this case, traveling from one contour to the next contour in a GTSP like manner, would result in laser head air moves that are (much) longer than the smallest distance between the two contours, thus increasing air movement costs.

Most likely however, an optimal tool path for a problem that contains both large and small contours will have both sub tool paths that resemble a tree and sub tool paths that resemble GTSP tours as depicted in figure 4(a). In figure 4(b), the contours are represented by a single vertex, which more clearly shows this *tree-like structure*. Figure 4(c) shows that by dividing the contours into seven groups, the tree-like structure can be modelled by an arborescence between the groups and by GTSP solutions within the groups. For the remainder of the text, we define a specific division of contours in groups as a *partitioning* and the resulting groups as *partitions*. The tool path problem can then be redefined as:

Finding a partitioning of contours, where a rooted directed minimum spanning tree is determined between the partitions and GTSPs are solved within the partitions in order to minimize the total path travel time.

Since any tool path can be represented by an array of elements, any tool path can alternatively be represented by a series of sub sequences of elements. And since the partitioning representation is basically a way of ordering these sub sequences, any tool path can be represented by the proposed representation.

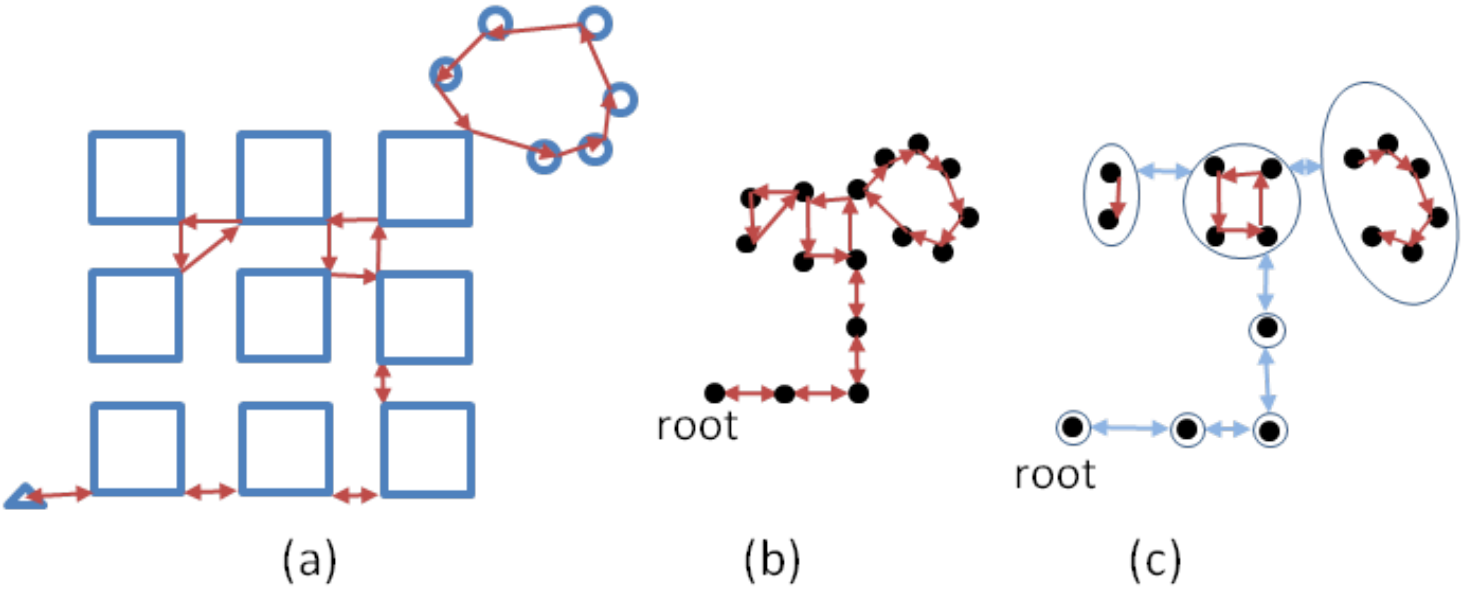

Figure 4. Tree-like structure in good tool paths and Arborescence-GTSP partitioning view

Representing the problem in this way allows efficient algorithms to solve the arborescence and GTSP subproblems. More specifically, the following data structure is utilized. The tree consists of one root partition that contains at least two contours (the laser head's starting location and the first contour to cut). Every contour can have multiple child partitions, but at any given node of the contour only a single partition can be attached. The node to which the child partition is attached is referred to as the parent node of the child partition. This structure allows improvement heuristics to more easily identify potential good local moves.

4. Improvement heuristic framework

We propose an improvement heuristic framework as follows:

(1) Determine & improve initial partitions^{PC}
  (a) Determine initial partitions^{PC}
  (b) Solve GTSPs within the partitions^{PC}
  (c) Repartition^{PC}
(2) Determine arborescence between the partitions
(3) Resolve arborescence
(4) Solve GTSPs within the partitions
(5) Repartition
(6) If stopping criterion is not reached, go to step 2. Else, go to step 7
(7) Restore feasibility^{PC}
(8) Improve partitions^{PC}
(9) Extract path

^{PC} designates that this step takes precedence constraints into account.
An initial partitioning of contours is determined first (1a) using a tool path construction heuristic. If the initial partitioning is the result of a construction heuristic that generates feasible paths, two improvement procedures are executed. The first improvement procedure involves solving the GTSP for every partition in the initial partitioning that does not contain precedence constraints (1b). The second improvement procedure is a set of repartition moves that takes precedence constraints into account (1c).

Then, while the stopping criterion is not reached, the algorithm determines an arborescence between the partitions (2,3), solves the GTSPs within the partitions (4), and executes a repartitioning phase (5). Steps 2, 3, 4, and 5 do not take precedence constraints into account.

If the stopping criterion is reached, a repair heuristic is executed to remove any infeasibilities that might have been introduced in steps 2-5 (7).

Given this feasible path, a set of repartition moves is executed that takes precedence constraints into account(8).

Additionally, every time entry/exit points are altered in contours, a new sub tool path is determined.

In the following subsections, each of these steps is discussed in more detail.

4.1 *Determining initial partitions*

Any tool path construction heuristic can be used to create an initial tool path from which the initial partitions can be determined. The heuristic used here is the PFr heuristic described in Dewil et al. (2014). Alternatively, our algorithm can also start from a single partition i.e. a GTSP solution; or with every contour in a separate partition i.e. a pure arborescence solution. These alternatives are used to evaluate how much the algorithm depends on the starting solution.

4.2 *Solving the GTSP within partitions*

Given a group of partitions, a GTSP can be solved within each partition. If a minimum spanning tree between partitions has already been determined, then the partition's parent node can be included in the GTSP sub problem to speed-up convergence. Noon and Bean (1991) proposed a method of transforming a GTSP into an asymmetric traveling salesman problem (ATSP) without introducing additional nodes. The resulting ATSP can be efficiently solved to near-optimality in a limited time span using a newer version of the Lin Kernighan heuristic (LKH) proposed by Helsgaun (2000, 2009). In our current implementation, precedence constraints are not taken into account in the GTSP solution, since initial observations showed that the tool paths obtained could

easily be restored to feasibility without significant changes in the objective function. During the improvement phase of the initial partitions, precedence constraints are indirectly taken into account by applying the LKH only on those partitions that do not contain any precedence constraints.

4.3 *Determining an arborescence between partitions*

Given a set of partitions, this step determines a rooted directed minimum spanning tree between the partitions. The partitions consist of contours which themselves consist of several nodes. However, there is no requirement that all arcs in the arborescence solution that are connected to a given partition have to be connected to the same node. In fact, connecting different partitions to the same node actually implies that one or more of these partitions must be merged.

A simple preprocessing phase identifying the cheapest connection between two partitions can transform the problem into a regular arborescence problem which can then be solved using the algorithm of Chu and Liu (1965) in O(E log V) with V being the number of vertices and E, the number of edges. To the best of our knowledge, no algorithm has been proposed that solves a precedence constrained rooted directed minimum spanning tree problem efficiently. Instead of developing such a procedure, we chose to ignore the precedence constraints and let the repair heuristic deal with restoring feasibility.

4.4 *Resolving arborescence*

Chu-Liu's algorithm results in a set of selected edges which connect all partitions. As can be seen in the top row of figure 5 there are four different ways that two partitions can be connected to one another which have to be resolved in such a way that the algorithm's structure of *"GTSP's within partitions and a RDMST between partitions"* is maintained. Both partitions originally consist of four contours and GTSP tours are defined for both partitions (blue arrows). The red arrows in the top row designate which nodes are selected to connect the two partitions in each scenario. These nodes are referred to as "connection nodes" in the following paragraphs.

In case (a) the selected connection node in the parent partition is not the current entry/exit node of its contour. The chosen connection node of the child partition is the entry/exit node its contour. This is the only case where no extra action needs to be undertaken to maintain the partition-tree structure. This case resolves to the following tool path. The first contour of the parent partition is cut completely after which the second contour is cut partly until the sub tool path reaches the selected pre-empt node where a pre-cut is placed and an air move is executed to the chosen entry node of the child partition. Then all contours of the child partition are cut according to the child partition's GTSP tour. Lastly, an air move is executed returning the laser head from the child partition to the parent partition where the rest of the second contour is cut followed by cutting the remaining contours in the parent partition.

In case (b) both the chosen connection node of the parent partition and the chosen connection node of the child partition are not entry/exit nodes of their respective contours. In this case, the contour of the chosen connection node in the child partition is placed in its own separate partition. This new partition becomes a child of the original parent partition and becomes the parent of the original child partition. This case resolves to the following tool path. The first contour of the parent partition is cut completely after which the second contour is cut until the sub tool path reaches the chosen connection node where a pre-cut is placed and an air move is executed to the chosen connection node of the child partition. This connection node is now the entry/exit node of its contour and is situated in its own partition. This contour is then cut until the sub tool path reaches the contour's original entry/exit node where a pre-cut is placed and an air move is executed to the next contour in the old GTSP path of the child partition (which has now effectively become the grand child partition). This GTSP tour is then executed after which the cutting head returns to the new child partition to finish cutting the contour. And then lastly, an air move is executed returning the laser head from the new child partition to the parent partition where the rest of the

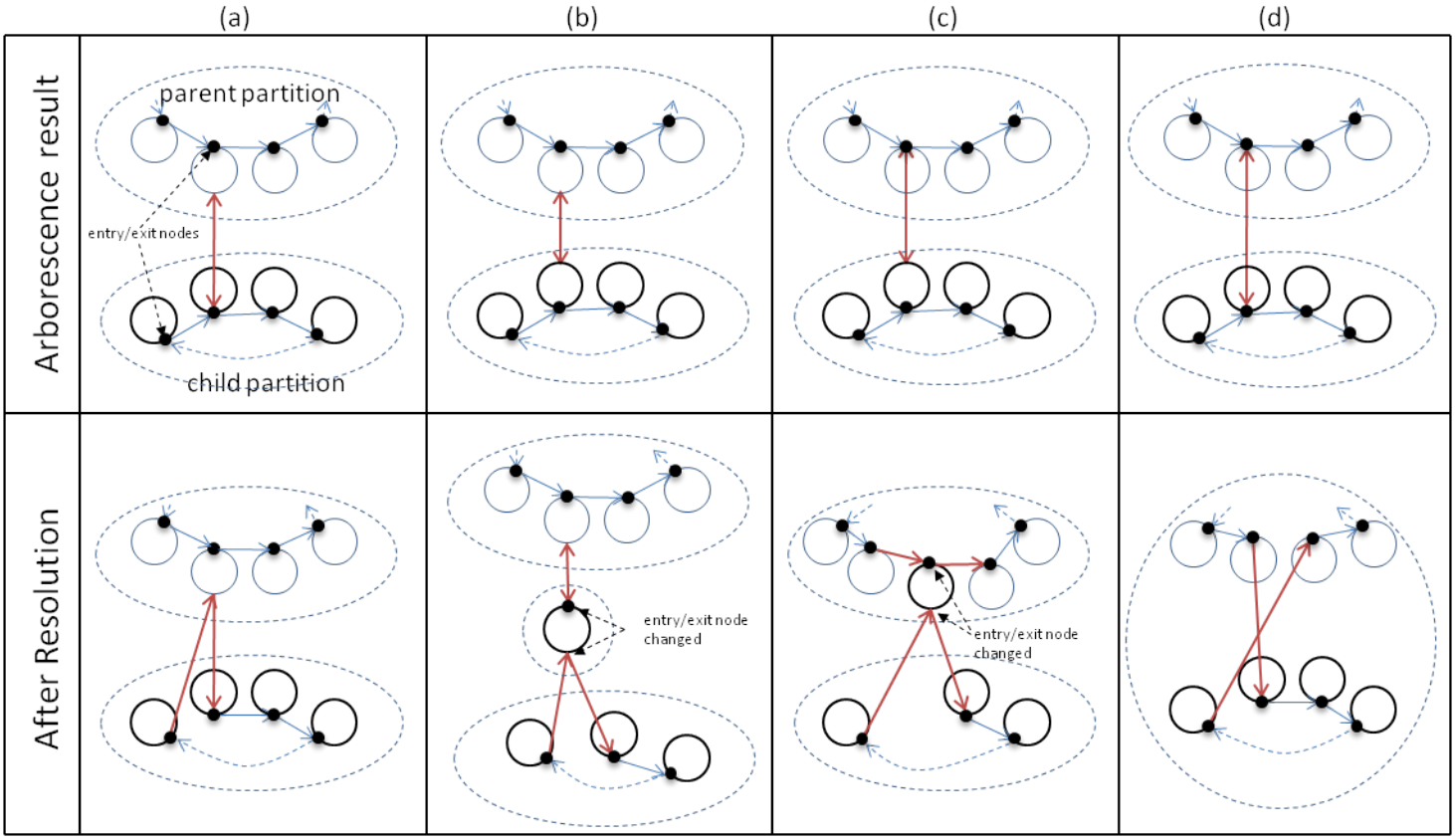

Figure 5. Minimum spanning tree based repartition moves

second contour is now cut followed by cutting the remaining contours in the parent partition.

In case (c) the chosen connection node of the parent partition is the entry/exit node of its contour and the chosen connection node of the child partition is not the entry/exit node of its contour. In this case, the chosen contour of the child partition is inserted in the parent partition. The child partition is attached to this contour and its original entry/exit node now becomes a pre-empt node.

In case (d) both the chosen connection node of the parent partition and the chosen connection node of the child partition are entry/exit nodes of their contours in their respective GTSP solutions. This configuration leads to a merging of both partitions into a single partition.

Similarly, if the solution of the arborescence would result in attaching multiple partitions to the same partition at the same node, a merger of the candidate child partitions is executed.

4.5 *Repartition*

The decision of which contours to repartition and where to place them is an important part of the algorithm. While the GTSP and arborescence solutions improve parts of the overall tool path and can be seen as intensification steps, the repartitioning moves change the overall structure of the tool path and thus can be seen as diversification steps. Several straightforward repartitioning moves have been implemented which are based on executing local moves on the tree-like structure. If a regular array-based path representation is used, these moves are basically 3-opt moves without sub path inversion (Croes (1958)). The advantage of using the tree-like structure is that only relevant moves are considered i.e. no moves are evaluated that would add expensive piercings.

The first set of repartitioning moves consists of two possible moves. The first move is simply the reattachment of a partition as a child of another partition. It is of course disallowed to attach a partition to any of its own descendants as this would result in two separate sub tours. The second move is more elaborate and is depicted in figure 6. This move is only considered when a partition

consists of single contour and contains at least one child that also consist of a single contour. In this case, a merger of the partition with its child partition is evaluated together with the evaluation of a re-attachment of the newly merged partition to another partition (or the same partition but to another node as in figure 6). If the child itself has single contour children, the merger and reattachment of the parent, child and grandchild is evaluated.

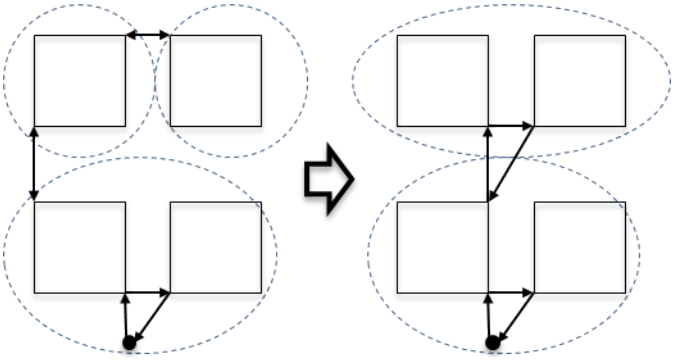

Figure 6. Combined re-attachment and merger of single pierce group partitions

The second set of repartitioning moves (as depicted in figure 7) consists of identifying sub sequences of pierce groups (pierce group j_{next} up to and including pierce group k) in a partition and also evaluating two possible moves.

The first move evaluates placing the chosen sequence in a separate partition and attaching this new partition to any other partition, including its original partition. For example, in figure 7(a), the sub sequence from pierce group j_{next} up to and including pierce group k is removed from partition 1. A new partition is created for this sub sequence and the new partition is added as a child to partition 1. In particular, it is added as a child to pierce group j.

The second move evaluates the insertion of the sub sequence in another partition, including its original partition (shown in figure 7(b)). For example, in figure 7(b), the sub sequence going from pierce group j_{next} up to and including pierce group k is removed from partition 1. This sub sequence is inserted in partition 2 in between pierce groups t and t_{next}.

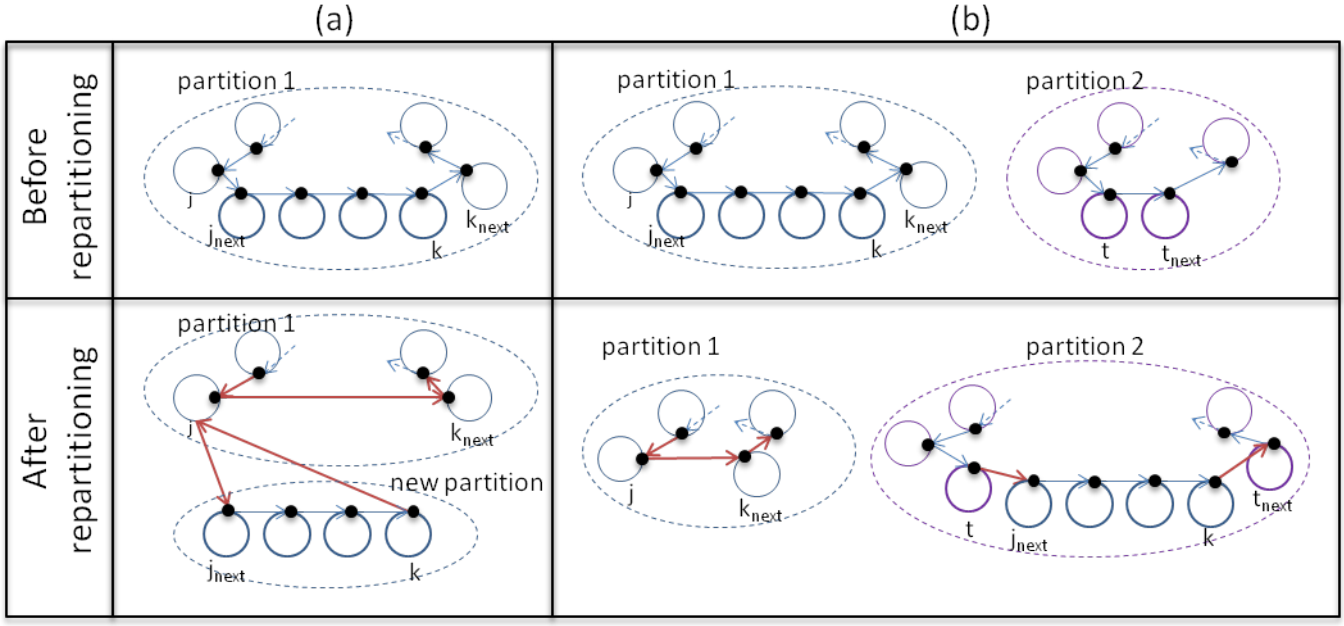

Figure 7. Moving sub sequences of pierce groups

We have implemented two versions for each set of repartitioning moves, one that accounts for precedence constraints and one that does not. The version that enforces precedence constraints, however, only functions on a feasible solution. As such, this version can only be applied as an improvement phase after the construction heuristic has determined an initial partitioning and once more at the end after the repair heuristic has restored feasibility.

4.6 *Restoring Feasibility*

Since precedence constraints are not enforced both in either the arborescence construction or the Lin-Kernighan heuristic, a repair heuristic was implemented to restore feasibility. The reparation phase consists of iterating over the generated tool path. If a contour is closed before all of its inner contours have been cut, then the entire contour is shifted towards the end of the path to a position just past the point where the last of its inner contours is cut. If this contour is part of a pierce group containing common cuts, the entire pierce group is shifted backwards. Following these repairs, a minor improvement phase is executed to determine new entry/exit nodes of the moved pierce groups and executes a repartitioning phase which takes precedence constraints into account.

4.7 *Building the sub tool path*

At several positions during the execution of the above procedures, new entry or exit points are determined for a pierce group. As such, a new sub tool path needs to be determined. This is trivial in the case of regular contours if precedence constraints are not taken into account[4]. It is however, more complicated in the case of common cut pierce groups. In the case of common cut pierce groups, the element insertion heuristic of Dewil et al. (2014) is used. This heuristic does not take other partitions into account to determine its inner air moves. Consider for example figure 8 where one common cut sub path needs to be determined in the vicinity of two partitions. If adjacent partitions would not be taken into account (figure 8a), the optimal sub path would select the two short air moves d-c and b-a. However this would entail that the adjacent partitions would have to be attached in a way that includes one short air move and one long air move (figure 8c). On the other hand, if adjacent partitions would be taken into account (figure 8b) air moves a-d and b-c would be selected which are two long air moves but which could be bridged by the adjacent partitions between these two nodes (figure 8d).

Several adaptations of the insertion based heuristic that attempted to take adjacent partition information into account were tested, but no positive difference in solution quality was noticed. Nonetheless, the idea of taking adjacent partition information into account when determining sub tool paths for common cut pierce groups remains valid and might be of interest for future research.

5. Computational results

The algorithm has been tested on the set of real-life benchmark instances of Dewil et al. (2014) and compared to tool paths generated by the PFr heuristic. The computational experiments were executed on an Intel Core i7-2630QM processor (2.00 GHz) with 6 GB of RAM. Table 1 shows the results for the 27 instances grouped in three sets according to the characteristics of the instances. Set 1 contains instances without common cuts with small parts and relatively large inter part distances, or parts with many inner contours that are evenly spread across the surface of the part. A good tool path in this set is expected to contain few pre-emptions, and GTSP solutions would yield high quality solutions. Set 2 contains instances with large parts and relatively small inter

[4]If precedence constraints would be actively included, care should be taken in choosing the cutting direction of the contours since this fixes the relative order of all nodes. If precedence relations exist between several child partitions of this pierce group, the relative order of the nodes does matter and could theoretically even cause a gridlock.

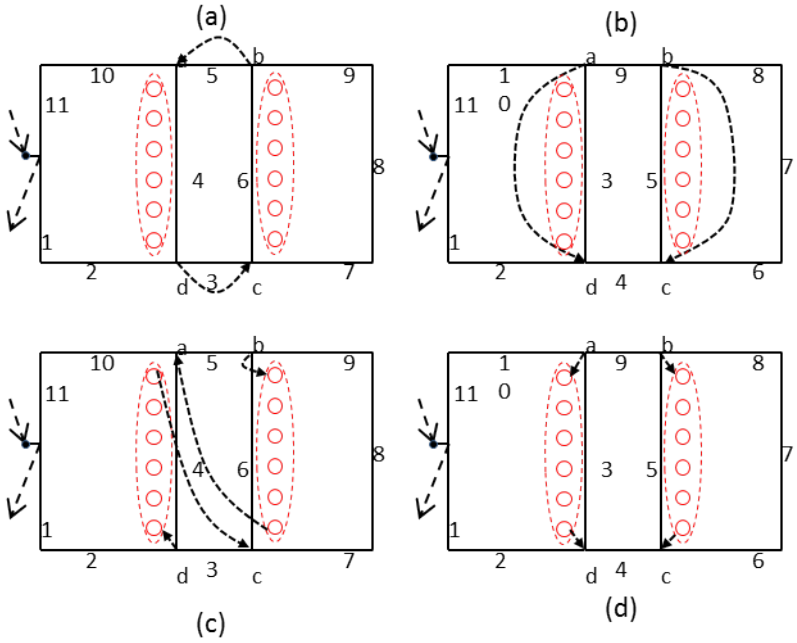

Figure 8. Selection of air moves based on adjacent partitions

part distances. It can be expected that good tool paths here would contain more pre-emptions. Lastly, set 3 contains common cuts with both large and small parts and large and small inter part distances.

The column *instance* contains the instance number. n represents the number of elements in the instance. *PFr* contains the objective function value of the construction heuristic in seconds. *GTSP* contains the result the Lin-Kernighan heuristic (obj - objective function values in seconds, %impr - percentage improvement over PFr) which does not take precedence constraints into account and as such does not necessarily contain a feasible tool path. However, since commercial CAD/CAM software only generates GTSP tool paths and the LKH is able to reach optimal solutions for instances of similar size than those considered here (Helsgaun (2014)), the GTSP solutions can be seen as a lower bound of commercial CAD/CAM software, especially for sets 1 and 2[5]. Column *PFr-I* contains the results (obj - objective function values in seconds, t - computational time in seconds, %impr - percentage improvement over PFr) of the construction heuristic together with the repartition/improvement phase that takes precedence constraints into account and the LKH GTSP heuristic for sub sequences of the initial path where no precedence constraints exist between the pierce groups. *F-NR* contains the results of the full algorithm with a stopping criterion of 10 non-improving iterations, not including the repair heuristic. *F-R* contains the results of the full algorithm including the repair heuristic. The difference between the F-NR and F-R results measures the efficacy of the repair heuristic.

From this table one can immediately observe that the repartitioning approach requires considerably more time than the construction heuristic. This is particularly evident in the case of instances 26 and 27 where the algorithm in every iteration resulted in one very large partition and a small number of small partitions, resulting in very large LKH calculation times. The results show that

[5]The computational experiments were performed using the default settings of the LKH of Helsgaun (2009). The settings specifically chosen for the *GTSP* described in Helsgaun (2014) were also tested. These settings resulted in an average 4.38% improvement over the PFr heuristic, compared to a 4.2% average improvement using the default settings. However, the required computation time nearly tripled. With the default settings, 7.7 hours were required to solve all instances while with the *GTSP* settings 22.2 hours were required to solve all instances. The increase is solely due to the longer LKH computation times i.e. the number of iterations (number of repartitioning calls) remained the same since the stopping criterion is: 10 non improving iterations and the changes that the new parameters introduced in the GTSP paths were insufficient to force extra iterations.

the construction plus improvement phase results in an improvement of on average 0.8%, while limiting the calculation time. Limited calculation times are important when tool paths need to be determined on the shop floor on the laser cutting machine itself. It is of less importance when work orders are prepared beforehand while the laser cutter is processing another order. If the calculation time is not a limiting issue, the full algorithm results on average in an (infeasible) improvement over the initial result of 5.2%.

This improvement reduces to 4.2% after feasibility is restored. A more detailed investigation showed that the feasibility restoration by itself reduces the improvement to on average 2.9%. Improving the entry/exit nodes of these solutions results in the 4.2% improvement. The improvement phase after restoring feasibility also executes a repartitioning taking precedence constraints into account. However it resulted in one executed move in all instances which increased the objective function by 0.003 seconds. Thus, one can conclude that after feasibility restoration, it suffices to only improve the entry/exit nodes of the pierce groups.

Table 1. Computational results for the full partitioning approach

	instance	n	PFr	GTSP		PFr-I			F-NR			F-R		
			obj.	obj.		obj.	t	%impr.	obj.	t	%impr.	obj.	t	%impr.
Set 1	10b	421	82.1	78.8	4.0%	81.8	0.6	0.3%	79.1	12.8	3.6%	79.1	13.5	3.6%
	12	1793	300.9	271.1	9.9%	296.6	2.1	1.4%	271.1	156.9	9.9%	279.7	159.0	7.0%
	13	289	96.5	84.8	12.0%	93.8	0.2	2.8%	85.3	10.7	11.6%	85.7	11.1	11.1%
	18	656	146.0	140.1	4.0%	146.0	0.0	0.0%	140.5	234.3	3.7%	142.6	234.9	2.3%
	19	1002	121.3	106.7	12.0%	121.0	0.2	0.2%	106.7	89.8	12.1%	110.5	90.2	8.9%
	20	2357	662.9	621.8	6.2%	662.0	6.4	0.1%	621.7	1205.8	6.2%	625.8	1211.2	5.6%
	21	529	268.0	263.6	1.7%	268.0	0.9	0.0%	263.6	11.1	1.6%	263.6	12.0	1.6%
	25	1424	295.9	273.3	7.6%	294.9	9.5	0.3%	273.0	238.7	7.7%	276.6	239.7	6.5%
	26	2810	394.8	367.2	7.0%	394.2	261.7	0.2%	367.7	18328.3	6.9%	370.5	18341.4	6.2%
	27	2194	308.6	288.9	6.4%	308.6	232.9	0.0%	289.6	7110.8	6.1%	291.3	7125.3	5.6%
	28	324	107.5	104.3	3.0%	107.5	0.0	0.0%	102.9	24.4	4.2%	103.1	24.5	4.1%
	30	145	76.9	70.2	8.7%	76.4	0.0	0.7%	70.6	0.5	8.2%	70.9	0.5	7.8%
			avg. impr.		**6.9%**			**0.5%**			**6.8%**			**5.9%**
Set 2	10	421	76.3	74.4	2.5%	75.1	0.1	1.6%	74.1	4.3	2.9%	74.1	4.4	2.9%
	17	145	9.7	10.0	-3.2%	9.7	0.0	0.0%	9.7	0.4	0.0%	9.7	0.4	0.0%
	29	121	66.2	66.9	-1.1%	66.2	0.0	0.0%	64.6	0.5	2.4%	64.6	0.6	2.4%
	34	91	50.6	49.61	1.9%	49.5	0.0	2.1%	48.3	1.6	4.4%	48.5	1.7	4.2%
	35	21	11.5	11.90	-3.3%	11.2	0.0	3.1%	11.2	0.0	3.1%	11.2	0.1	3.1%
	36	61	33.8	33.88	-0.4%	33.0	0.0	2.3%	32.2	0.6	4.8%	32.3	0.7	4.4%
	37	31	17.5	18.54	-5.8%	16.5	0.0	5.6%	16.3	0.1	7.1%	16.3	0.2	7.1%
	38	237	74.2	74.66	-0.7%	73.8	0.0	0.5%	73.8	0.3	0.5%	73.8	0.4	0.5%
			avg. impr.		**-1.3%**			**1.9%**			**3.2%**			**3.1%**
Set 3	11	745	178.1	172.2	3.3%	178.0	0.6	0.1%	171.8	34.7	3.5%	172.4	35.0	3.2%
	14	230	44.6	53.1	-19.0%	44.3	0.0	0.7%	43.7	9.5	2.1%	44.0	9.9	1.3%
	15	253	62.9	68.0	-8.2%	62.9	0.0	0.0%	59.2	18.6	5.9%	61.5	18.7	2.2%
	16	303	67.9	72.1	-6.2%	67.9	0.0	0.0%	62.1	9.5	8.5%	64.4	9.7	5.1%
	16b	123	52.4	55.4	-5.7%	52.2	0.0	0.4%	48.2	0.0	8.0%	50.8	0.1	3.1%
	22	385	191.6	196.0	-2.3%	191.6	0.3	0.0%	183.9	1.8	4.0%	187.1	2.0	2.3%
	23	841	252.7	263.0	-4.1%	252.7	1.2	0.0%	246.2	15.5	2.6%	248.5	15.6	1.6%
			avg. impr.		**-6.0%**			**0.9%**			**4.5%**			**2.9%**
			avg. impr. all instances		**1.1%**			**0.8%**			**5.2%**			**4.2%**

These results further show that in the Set 1 instances, the GTSP solutions, which set the lower bound on commercial CAD/CAM software, yield an average improvement of 6.9% over the PFr construction heuristic. In comparison, our solution approach guarantees feasible solutions and yields on average an improvement of 5.9% over the PFr construction heuristic, resulting in a difference of only 1%. For both the Set 2 and the Set 3 instances, the GTSP approach yields on average worse results than the construction heuristic. The differences between the GTSP approach and the proposed solution approach are on average 4.4% and 8.2% for the Set 2 and the Set 3 instances respectively, in favor of the proposed solution approach.

Table 2. Using different starting partitions

	instance	n	Constr	PFr		SP		GTSP	
				obj.	%impr	obj.	%impr	obj.	%impr
Set 1	10b	421	82.1	79.1	3.6%	**78.8**	4.0%	79.4	3.3%
	12	1793	300.9	**279.7**	7.0%	280.2	6.9%	280.1	6.9%
	13	289	96.5	85.7	11.1%	**84.8**	12.1%	85.6	11.3%
	18	656	146.0	**142.6**	2.3%	142.7	2.3%	142.8	2.2%
	19	1002	121.3	**110.5**	8.9%	111.9	7.7%	113.0	6.8%
	20	2357	662.9	625.8	5.6%	626.6	5.5%	**623.5**	5.9%
	21	529	268.0	263.6	1.6%	**263.6**	1.7%	263.6	1.7%
	25	1424	295.9	276.6	6.5%	**275.4**	6.9%	277.1	6.3%
	26	2810	394.8	370.5	6.2%	**367.0**	7.1%	372.1	5.7%
	27	2194	308.6	291.3	5.6%	**290.6**	5.8%	291.1	5.7%
	28	324	107.5	**103.1**	4.1%	103.9	3.3%	104.0	3.2%
	30	145	76.9	70.9	7.8%	**72.5**	5.8%	70.8	7.9%
Set 2	10	421	76.3	**74.1**	2.9%	74.3	2.7%	**74.1**	2.9%
	17	145	9.7	**9.7**	0.0%	**9.7**	0.0%	**9.7**	0.3%
	29	121	66.2	**64.6**	2.4%	65.3	1.2%	65.8	0.5%
	34	91	50.6	**48.5**	4.2%	49.4	2.2%	50.0	1.0%
	35	21	11.5	**11.2**	3.1%	11.3	1.6%	11.7	-1.3%
	36	61	33.8	**32.3**	4.4%	33.3	1.2%	33.9	-0.6%
	37	31	17.5	**16.3**	7.1%	17.3	1.3%	19.7	-12.7%
	38	237	74.2	**73.8**	0.5%	74.2	0.0%	74.8	-0.9%
Set 3	11	745	178.1	**172.4**	3.2%	173.4	2.6%	176.6	0.8%
	14	230	44.6	**44.0**	1.3%	44.9	-0.7%	45.5	-2.0%
	15	253	62.9	**61.5**	2.2%	62.7	0.2%	63.0	-0.2%
	16	303	67.9	64.4	5.1%	65.7	3.2%	**64.2**	5.4%
	16b	123	52.4	**50.8**	3.1%	51.8	1.2%	52.1	0.6%
	22	385	191.6	**187.1**	2.3%	190.9	0.4%	191.7	-0.1%
	23	841	252.7	**248.5**	1.6%	252.3	0.2%	251.1	0.6%
			avg. impr.		**4.2%**		**3.2%**		**2.3%**

Table 2 shows the results for the same 27 instances using different starting partitions. The *Constr.* column contains the results of the construction heuristic. The *PFr* column contains the results where the algorithm starts from the construction heuristic, which corresponds to the F-R results in Table 1. *SP* contains the results where the algorithm starts with each pierce group in its own partition. Oppositely, the *GTSP* contains the result where the algorithm starts with all pierce groups in a single partition.

From these results, we can conclude that the starting solution still has an effect on the end result. However, it can be seen that no starting solution can guarantee to find the best result (shown in bold) in all cases. Furthermore, the quality of the starting solution doesn't necessarily mean that the end result will be of higher quality than the ones obtained with other starting solutions. For instance, starting from a GTSP solution results in only 1 best final solution out of 12 set 1 instances, while the GTSP starting solutions are indeed of higher quality than the PFr and SP starting solutions for these instances. The average improvements over the construction heuristic solutions are 4.2%, 3.2% and 2.3% for respectively the PFr, SP, and GTSP starting solutions. Deeper investigation showed that in all experiments more partition mergers in the arborescence resolution phase are executed than sub sequence ejection moves in the repartitioning phase. This can explain the relatively poor performance of using a complete GTSP starting solution. Investigation of additional ejection moves or a meta-heuristic approach to escape local optima is suggested to

improve the diversifying nature of the repartitioning phase.

Table 3. Impact of individual components

	instance	n	C-FR %impr	C-MP %impr	C-MSP %impr	C-LKH-FR %impr
Set 1	10b	421	0.3%	0.0%	0.3%	**2.2%**
	12	1793	1.4%	0.0%	0.0%	**1.5%**
	13	289	2.8%	0.0%	0.0%	**3.1%**
	18	656	0.0%	0.0%	0.0%	**0.9%**
	19	1002	0.2%	0.0%	0.2%	**2.7%**
	20	2357	0.1%	0.1%	0.0%	**2.4%**
	21	529	0.0%	0.0%	0.0%	**1.6%**
	25	1424	0.3%	0.3%	0.0%	**2.9%**
	26	2810	0.2%	0.2%	0.0%	**5.0%**
	27	2194	0.0%	0.0%	0.0%	**2.2%**
	28	324	0.0%	0.0%	0.0%	**0.6%**
	30	145	0.7%	0.9%	0.1%	**2.4%**
Set 2	10	421	1.6%	0.0%	0.7%	**1.9%**
	17	145	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	29	121	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	34	91	**2.1%**	0.8%	**2.1%**	**2.1%**
	35	21	**3.1%**	3.1%	**3.1%**	**3.1%**
	36	61	**2.3%**	1.1%	**2.3%**	**2.3%**
	37	31	**5.6%**	2.0%	**5.6%**	**5.6%**
	38	237	**0.5%**	**0.5%**	**0.5%**	**0.5%**
Set 3	11	745	0.1%	0.0%	0.0%	**1.6%**
	14	230	0.7%	0.0%	0.7%	**1.6%**
	15	253	0.0%	0.0%	0.0%	**2.2%**
	16	303	0.0%	0.0%	0.0%	**3.5%**
	16b	123	0.4%	0.1%	0.0%	**1.0%**
	22	385	0.0%	0.0%	0.0%	**1.9%**
	23	841	0.0%	0.3%	0.0%	**0.9%**
			0.8%	**0.3%**	**0.6%**	**2.1%**

Table 3 shows the impact of the individual components of the algorithm during the construction/improvement phase. Column *C-FR* contains the results for a full repartitioning/improvement after the PFr construction heuristic. *C-MP* only allows the first set of repartition moves: the movement of partitions and possible movement and merger of partitions containing a single pierce group with their single pierce group child partitions. *C-MSP* only allows the second set of repartition moves i.e. the movement of sub sequences of pierce groups into a separate partition or the insertion of sub sequences of pierce groups into another partition. And lastly, *C-LKH-FR* contains the construction heuristic, the LKH improvement step and the full repartitioning/improvement phase. The results show that on average the full repartitioning improves the initial solution by 0.8% whereas only moving/merging partitions and only moving sub paths both result in an improvement of only 0.3% and 0.6%. But the largest improvement is gained by including the optimization of sub sequences by the Lin-Kernighan heuristic, which increases the average improvement over the initial solution to 2.1%.

The final solution structures are heavily dependent on the instance type. The solution approach results for Set 1 instances in final solutions with a limited number of partitions, typically between 1 and 5 partitions. In the instances where the final solution contains multiple partitions, there typ-

ically is one large partition containing most pierce groups and several smaller partitions containing few pierce groups. The final solutions in the Set 2 and Set 3 instances typically contained many partitions of differing sizes. The number of partitions in the final solutions for all three Sets were nearly the same regardless of the starting heuristic.

Unfortunately, the inclusion of the Lin-Kernighan heuristic is computationally expensive. As can be seen from figure 9, the GTSP optimization takes on average 60.22% of the total computation time. But the LKH execution can reach as high as 97.54% and 83.38% of the time for instances 26 and 27 respectively. The repartitioning step is the second most computationally expensive step requiring 29.34% of time, followed by the repair heuristic (5.47%), the arborescence construction and resolution (3.89%), the path build and NC code generation (0.65%) and the construction phase (0.47%).

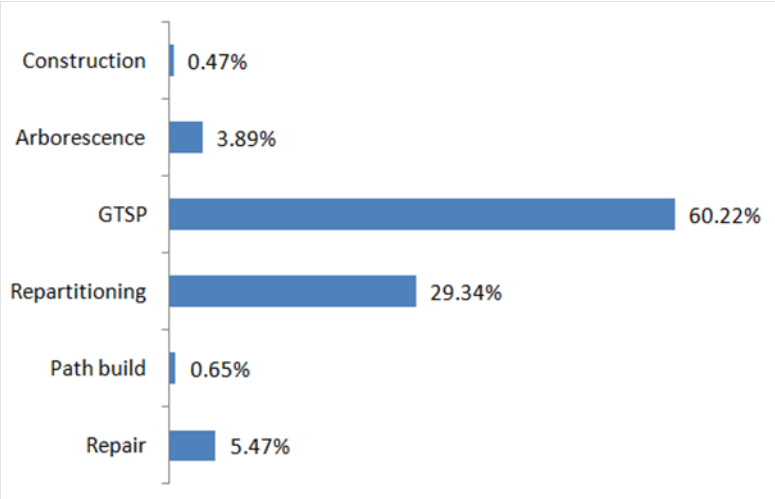

Figure 9. Time expenditure algorithm

6. Conclusions

In this paper a novel way of representing tool paths for laser cutting machines is presented together with a hybrid algorithm exploiting this structure. This algorithm consists of consecutively assigning pierce groups to partitions where generalized traveling salesman problems are solved within the partitions and a rooted directed minimum spanning tree problem is solved across the partitions. In the first iteration, the assignment of pierce groups to partitions is executed through a tool path construction heuristic and in subsequent steps, this assignment occurs through local search moves on the partition-tree. Computational results are reported for a set of 27 instances with between 21 and 2810 elements. The hybrid partition-tree approach is able to improve upon the construction heuristic by on average 4.2% with a maximum improvement of 11.1%. Considering the solutions found from different starting points, the new approach was able to establish new benchmarks for all problems in this data set (see Table 2).

The repartition moves implemented are based on local moves on the partition-tree representation, but further research might be useful to add a component to implement a more systematic search of the solution space.

References

Castelino, K., D'Souza, R., Wright, P., 2003. Toolpath optimization for minimizing airtime during machining. Journal of Manufacturing Systems 22 (3), 173–180.

Chu, Y. J., Liu, T. H., 1965. On the shortest arborescence of a directed graph. Science Sinica 14, 1396–1400.
Croes, G. A., 1958. A method for Solving Traveling Salesman Problems. Operations Research 6 (6), 791–812.
Dewil, R., Vansteenwegen, P., Cattrysse, D., Mar. 2011. Cutting Path Optimization Using Tabu Search. Key Engineering Materials 473, 739–748.
Dewil, R., Vansteenwegen, P., Cattrysse, D., 2012. Generating tool paths for laser cutting machines. In: ORBEL 2012. Brussels, pp. 2–3.
Dewil, R., Vansteenwegen, P., Cattrysse, D., Mar. 2014. Construction heuristics for generating tool paths for laser cutters. International Journal of Production Research, 1–20.
Gambardella, L. M., Dorigo, M., 2000. An ant colony system hybridized with a new local search for the sequential ordering problem. INFORMS Journal on Computing 12 (3), 237–255.
Garfinkel, R. S., Webb, I. R., Oct. 1999. On crossings, the Crossing Postman Problem, and the Rural Postman Problem. Networks 34 (3), 173–180.
Glover, F., Laguna, M., Jan. 1999. Tabu Search. Vol. 50. Kluwer Academic Publishers, Boston.
Han, G.-c., Na, S.-j., 1999a. A Study on Torch Path Planning in Laser Cutting Processes Part 1: Calculation of heat Flow in Contour Laser Beam Cutting. Journal of Manufacturing Processes I (1), 54–61.
Han, G.-c., Na, S.-j., 1999b. A Study on Torch Path Planning in Laser Cutting Processes Part 2 : Cutting Path Optimization. Journal of Manufacturing Processes 1 (1), 62–70.
Helsgaun, K., Oct. 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research 126 (1), 106–130.
Helsgaun, K., Jul. 2009. General k-opt submoves for the LinKernighan TSP heuristic. Mathematical Programming Computation 1 (2-3), 119–163.
Helsgaun, K., 2014. Solving the Equality Generalized Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm. Tech. Rep. December 2013, Roskilde University, Roskilde.
Hoeft, J., Palekar, U. S., Sep. 1997. Heuristics for the plate-cutting traveling salesman problem. IIE Transactions 29 (9), 719–731.
Imahori, S., Kushiya, M., Nakashima, T., Sugihara, K., Sep. 2008. Generation of cutter paths for hard material in wire EDM. Journal of Materials Processing Technology 206 (1-3), 453–461.
Jing, Y., Zhige, C., 2013. An Optimized Algorithm of Numberical Cutting-Path Control in Garment Manufacturing. Advanced Materials Research 796, 454–457.
Kim, Y., Gotoh, K., Toyosada, M., Jul. 2004. Global cutting-path optimization considering the minimum heat effect with microgenetic algorithms. Journal of Marine Science and Technology 9 (2), 70–79.
Kruskal, J. B., 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society 7 (1), 48–50.
Lee, M.-K., Kwon, K.-B., Dec. 2006. Cutting path optimization in CNC cutting processes using a two-step genetic algorithm. International Journal of Production Research 44 (24), 5307–5326.
Moreira, L., Oliveira, J., Gomes, A., Ferreira, J., Nov. 2007. Heuristics for a dynamic rural postman problem. Computers & Operations Research 34 (11), 3281–3294.
Noon, C., Bean, J., 1991. An efficient transformation of the generalized traveling salesmen problem. INFOR 31 (1), 1993.
Raggenbass, A., Reissner, J., 1989. Stamping - Laser Combination in Sheet Processing. Annals of the CIRP 38 (1), 291–294.
Raggenbass, a., Reissner, J., Jan. 1991. Automatic Generation of NC Production Plans in Stamping and Laser Cutting. CIRP Annals - Manufacturing Technology 40 (1), 247–250.
Srivastava, S. S., Kumar, S., Carg, R. C., Sen, P., 1969. Generalized traveling salesman problem through n sets of nodes. CORS 7, 97–101.
Tarjan, R. E., 1977. Finding optimum branchings. Networks 7 (1), 25–35.
Vaupotic, B., Kovacic, M., Ficko, M., Balic, J., 2006. Concept of automatic programming of NC machine for metal plate cutting by genetic algorithm method. Journal of Achievements in Materials and Manufacturing Engineering 14 (1), 131–139.
Veeramani, S., Kumar, D., Jul. 1998. Optimization of the nibbling operation on an NC turret punch press. International Journal of Production Research 36 (7), 1901–1916.
Wang, G. G., Xie, S. Q., Jun. 2005. Optimal process planning for a combined punch-and-laser cutting machine using ant colony optimization. International Journal of Production Research 43 (11), 2195–2216.
Xie, S. Q., Gan, J., Wang, G. G., Vn, C., 2009. Optimal process planning for compound laser cutting and punch using Genetic Algorithms. International Journal of Mechatronics and Manufacturing Systems 2 (1/2), 20–38.

Xie, S. Q., Tu, Y. L., Liu, J. Q., Zhou, Z. D., Jan. 2001. Integrated and concurrent approach for compound sheet metal cutting and punching. International Journal of Production Research 39 (6), 1095–1112.

Yang, W. B., Zhao, Y. W., Jie, J., Wang, W. L., Mar. 2010. An Effective Algorithm for Tool-Path Airtime Optimization during Leather Cutting. Advanced Materials Research 102, 373–377.