Report on Reverse-Engineering Automatic Bend Sequencing Logic for Salvagnini Panel Benders (Phase 3)

1. Executive Summary

This report constitutes the comprehensive deliverable for Phase 3 of the reverse-engineering project focused on deriving the "Optimal Bend Sequencing Algorithm" for Salvagnini P4 Panel Benders. The primary objective is to synthesize a robust, algorithmic framework capable of transforming a parsed Boundary Representation (B-Rep) graph—where nodes represent faces and edges represent bends—into a valid, time-optimized linked list of MachineAction objects. This sequencing logic serves as the computational core for Computer-Aided Process Planning (CAPP) systems analogous to the proprietary STREAMBEND software.

The research leverages a deep analysis of the Salvagnini P4's unique kinematic architecture, specifically distinguishing it from traditional press brake operations. The P4 system operates on a horizontal manipulation plane, utilizing a universal bending tool set that requires no physical changeover for varying geometries, coupled with an Automatic Blankholder (ABA) that adjusts tool lengths in "masked time".[1] These physical attributes fundamentally alter the cost functions used in algorithmic path planning. Unlike press brakes, where tool changes are binary and costly "stop" events, the P4's ABA adjustments are conditional costs that can be absorbed into the duration of manipulator rotations.

The core of this report proposes a **Heuristic State-Space Search** approach, specifically prioritizing the *A (A-Star) Algorithm** over Greedy or Genetic alternatives. We define a specific cost function $f(n) = g(n) + h(n)$ that encodes the machine's physics: $g(n)$ accumulates the real-time execution cost (prioritizing masked operations), while $h(n)$ utilizes geometric reasoning to penalize state transitions that would require future inefficiencies, such as unnecessary rotations or unmasked tool adjustments. The heuristic weights are derived from the operational reality that manipulator rotation is the dominant cycle-time driver ($w_{rot} >> w_{bend}$), while ABA adjustments hold a medium, conditional weight (w_{aba}).

The document provides an exhaustive technical breakdown of:

1. **Machine Kinematics & Constraints:** Formalizing the P4's physical limitations—specifically manipulator gripping zones and ABA segmentation—into rigid algorithmic rules.

2. **Geometric Reasoning:** Utilizing the B-Rep graph for precedence determination and collision avoidance using Hierarchical Bounding Volumes (AABB/OBB).
3. **Algorithmic Core:** Detailed pseudo-code for the A* solver, including the derivation of admissible heuristics for the specific topology of sheet metal parts.
4. **Output Generation:** The translation of the optimal search path into a linear MachineAction linked list, suitable for low-level machine control.

The findings confirm that a backward-planning (unfolding) strategy, driven by a collision-aware A* solver, offers the most reliable method for generating collision-free, minimum-time sequences for the P4 architecture.[3]

---

2. Introduction to Automated Panel Bending and CAPP

The transition from manual press brakes to automated panel benders represents a paradigm shift in sheet metal fabrication, moving from skill-dependent batch processing to algorithmic, flexible manufacturing systems (FMS). The Salvagnini P4 stands as the archetype of this technology, necessitating a fundamentally different approach to Computer-Aided Process Planning (CAPP).

2.1 The Operational Context: Salvagnini P4 Architecture

To derive an accurate sequencing algorithm, one must first mathematically model the physical reality of the target machine. The P4 is not merely a bending machine; it is an integrated manipulation and forming cell.

2.1.1 The Manipulator

The manipulator is the primary mechanism for feeding, rotating, and holding the sheet metal. It dictates the feasibility of any given bend sequence based on the remaining unbent surface area available for gripping.

- **Kinematics:** The manipulator operates in the XY horizontal plane with rotation θ (yaw). It clamps the sheet along the Z-axis.
- **Precision:** The continuous rotator ensures an angular resolution of 0.01°, allowing for precise correction of material behavior but imposing a time penalty for any angular change $\Delta\theta$.[5]
- **Constraint Implication:** Unlike a human operator who can grip a flange or a corner, the automatic manipulator requires a stable, flat "island" of material. As the bending sequence progresses, this flat area diminishes. The algorithm must therefore verify that for every state S_i in the sequence, there exists a valid face $F_{grip} \in Faces_{unbent}$

such that $Area(F_{grip}) \geq Area_{min\_clamp}$.

2.1.2 The Automatic Blankholder (ABA)

The ABA is the defining feature that enables "Batch One" production.[1] It consists of segmented upper tools that automatically adjust their combined width to match the length of the bend line.

- **Masked Time:** The crucial algorithmic feature of the ABA is its ability to adjust *during* the manipulator's positioning cycle. If the time required to rotate the sheet (t_{rot}) exceeds the time required to adjust the tool (t_{aba}), the effective cost of the tool change is zero.[2]
- **Constraint Implication:** The sequencing algorithm must exploit this by coupling major rotations with major tool length changes, effectively hiding the setup time.

2.1.3 The Bending Blades

The P4 uses an upper and lower blade configuration that moves in an interpolated path (Rolling Mode) to fold the metal against the blankholder.[5]

- **Polarity:** The machine can perform positive (UP) and negative (DOWN) bends sequentially without flipping the sheet.
- **Constraint Implication:** This eliminates the "Flip" cost function found in press brake algorithms but introduces a "Clearance" constraint. The blades must have a clear path to the bend line, requiring rigorous collision checking against previously bent flanges.

2.2 The Problem of Bend Sequencing

The "Bend Sequencing Problem" (BSP) is a classic NP-Hard combinatorial optimization problem.[7] For a part with n bends, there are $n!$ possible sequences. For a typical panel with 20 bends, $20! \approx 2.4 \times 10^{18}$ permutations. Most of these permutations are physically impossible due to:

1. **Collisions:** Bending flange A might block the tool from accessing flange B.
2. **Gripping Failures:** Bending all central surfaces might leave nowhere for the manipulator to hold the part for the final peripheral bends.

Therefore, the objective is not just to find *any* valid sequence, but to find the *optimal* sequence that minimizes cycle time (T_{cyc}). In the context of the P4, T_{cyc} is dominated by manipulator rotation and unmasked ABA adjustments, rather than the rapid stroke of the bending blades.[1]

---

3. Geometric Representation and Data Structures

The foundation of any CAPP system is the robust representation of the part geometry. For this project, we utilize a parsed B-Rep (Boundary Representation) Graph.

3.1 The B-Rep Graph Topology

The input data is structured as a graph $G = (V, E)$, where the topology of the sheet metal part is encoded in nodes and edges.

- **Nodes (V):** Represent the planar faces of the sheet. In a flat state, all nodes are coplanar. In a folded state, they occupy 3D space.
  - $V = \{f_1, f_2, ..., f_m\}$
  - Attributes: Area, Centroid, NormalVector (in folded state).
- **Edges (E):** Represent the connections between faces.
  - $E = \{e_1, e_2, ..., e_k\}$
  - Attributes:
    - Type: {Bend, Cut, Hem}
    - Angle: The target deformation angle (e.g., 90°, -45°).
    - Direction: The polarity relative to the top surface (Up/Down).
    - Length: The physical length of the bend line (critical for ABA input).
    - Radius: The internal bend radius.

3.2 Pre-Processing: The Precedence Graph

Before identifying the sequence, we must identify the hard constraints imposed by geometry. We generate a **Precedence Graph** (or Dependency Graph) P, which is a Directed Acyclic Graph (DAG) where a directed edge $A \rightarrow B$ implies that bend A must be performed *before* bend B (or *after* in an unfolding logic).[3]

The construction of P involves three types of geometric reasoning:

1. **Corner Overlap Constraints:**
   - If bend b_i and bend b_j share a vertex (a corner), the physical relief (miter, overlap, or tear) dictates the order. For example, in a closed corner, the flanges often must be bent simultaneously or in a strict overlap order to avoid tearing.
2. **Enclosure Constraints (C-Channels):**

- If bend b_i is inside the U-profile created by bends b_j and b_k, b_i typically must be bent before the enclosing walls b_j, b_k are formed. In an unfolding search, this means the enclosing walls must be unbent *first*.

3. **Tool Access Constraints (Ray Casting):**
   - For every bend b_i, we simulate the tool geometry approaching the bend line. We cast rays from the tool path vector. If the rays intersect any face f_k that is created by bend b_j, then b_j is an obstruction.
   - Rule: If b_j blocks b_i, then b_i must precede b_j in the bending sequence (or b_j precedes b_i in unfolding).

The result is a set of constraints that drastically prunes the search space. If the Precedence Graph dictates $b_1 \rightarrow b_2 \rightarrow b_3$, the solver does not need to explore the $3! = 6$ permutations of these bends; it only considers the single valid order.

---

4. Constraint Modeling and Kinematics

The "intelligence" of the sequencing algorithm resides in how accurately it models the machine's constraints. We move beyond simple geometric collisions to model the specific physics of the Salvagnini P4.

4.1 Manipulator Constraints: The "Dead Zone"

The P4 manipulator is a physical object with mass and volume. It cannot grasp empty space, nor can it grasp a bent flange (typically).

Formalization:

Let M be the footprint of the manipulator clamps. Let S_i be the state of the part at step i. Let F_{flat} be the union of all faces in S_i that are coplanar with the manipulator reference plane ($Z = 0$).

The constraint function $C_{grip}(S_i)$ returns TRUE if and only if:

1. $\exists P \subset F_{flat}$ such that $M \subseteq P$ (The manipulator footprint fits entirely within the flat area).

2. The friction center of M is close enough to the Center of Mass (CoM) of the part to prevent torque slippage during rotation.
3. The path of M from the loading position to P is collision-free.

This constraint is dynamic. As the part is bent (or unbent), faces leave (or enter) the F_{flat} set. The algorithm must ensure that a valid grip exists *continuously* or that a re-gripping operation is feasible (though re-gripping is expensive and should be minimized).

4.2 ABA Masked Time Calculus

The Automatic Blankholder Adjustment (ABA) logic is the differentiator for optimizing P4 cycle times. We model this as a "Conditional Cost."

Let L_{curr} be the current ABA length and L_{next} be the required length for the next bend.

Let θ_{curr} be the current sheet orientation and θ_{next} be the required orientation.

The time to adjust the ABA, T_{aba}, is:

$$T_{aba} = \frac{|L_{next} - L_{curr}|}{V_{aba}}$$

where V_{aba} is the velocity of the ABA steppers.

The time to rotate the sheet, T_{rot}, is:

$$T_{rot} = \frac{|\theta_{next} - \theta_{curr}|}{V_{rot}}$$

where V_{rot} is the rotational velocity of the manipulator.

The **Effective Cost** of the operation, $Cost_{op}$, is not simply the sum. It is the maximum of the parallel operations:

$$Cost_{op} = \max(T_{aba}, T_{rot}) + T_{bend}$$

However, from an *optimization* perspective, we view the ABA change as having a "Shadow

Cost":

$$Cost_{shadow} = \max(0, T_{aba} - T_{rot})$$

If $T_{aba} < T_{rot}$, the shadow cost is 0. The ABA change is "free" (Masked).

If $T_{aba} > T_{rot}$, the shadow cost is positive. The machine is waiting for the tool.

Insight: The algorithm should greedily pair large rotations with large ABA changes. A sequence that performs all 500mm bends, then rotates, then performs all 200mm bends is superior to one that alternates 500mm (0°) $\rightarrow$ 200mm (0°) $\rightarrow$ 500mm (90°).

4.3 Collision Constraints (Hierarchical)

To ensure the proposed sequence is physically valid, we employ a hierarchical collision detection strategy suitable for the iterative nature of the search.[9]

1. **Level 1: AABB (Axis-Aligned Bounding Box):**
    - Fastest check. We compute the AABB of the folded part state and the AABB of the machine throat/tooling.
    - If Intersection(AABB_Part, AABB_Machine) == False, the state is safe.
2. **Level 2: OBB (Oriented Bounding Box):**
    - If AABBs intersect, we compute OBBs, which are tighter fits for rotated parts. This reduces false positives.
3. **Level 3: B-Rep / Mesh Intersection:**
    - If OBBs intersect, we perform a computationally expensive face-to-face intersection test. This is only done when absolutely necessary.

This hierarchy allows the A* search to evaluate thousands of states per second by rejecting the majority of collisions at Level 1 or 2.[11]

---

5. Algorithmic Framework: Heuristic Search

While Genetic Algorithms (GA) and Simulated Annealing (SA) are discussed in literature for CAPP [12], *A (A-Star)** is generally preferred for bend sequencing because:

1. **Completeness:** If a solution exists, A* will find it.
2. **Optimality:** With an admissible heuristic, A* guarantees the lowest cost path.
3. **Determinism:** Unlike GA, A* produces the same output for the same input, which is crucial for industrial reliability.

5.1 Search Strategy: Backward Planning (Unfolding)

We adopt a **Backward Planning** strategy.[4] The search begins at the *Folded State* (the finished part) and attempts to find a sequence of "Unbends" to reach the *Flat State*.

- **Rationale:** The folded state is the most constrained. Collisions are most likely here. By starting at the end, we immediately identify the "locking" bends—those that *must* be done last (and thus unbent first). This effectively uses the geometric constraints to prune the search tree early.

5.2 The Cost Function $f(n)$

The core of the A* algorithm is minimizing $f(n) = g(n) + h(n)$.

5.2.1 $g(n)$: The Accumulated Path Cost

$g(n)$ measures the "cost so far" to reach the current state from the start (folded) state.

$$g(n) = \sum_{i=1}^{k} C(action_i)$$

Where $C(action_i)$ is determined by the machine constraints defined in Section 4.

Action Type	Parameter	Cost Weight (Arbitrary Units)	Rationale
Rotation	90^c	100	High mechanical inertia; slowest operation.
Rotation	180^c	120	Slightly higher than 90, but setup is dominant.
Small Rotate	$<$	20	Correction moves are faster but still non-zero.
ABA Wait	Unmasked	50	Penalty for

			stopping production to resize tool.
ABA Masked	Masked	0	**Critical Optimization**: Zero cost if hidden.
Bend	Standard	10	The irreducible baseline cost of the process.
Flip	N/A	∞	P4 does not flip.

5.2.2 $h(n)$: The Heuristic Function

$h(n)$ estimates the remaining cost to reach the flat pattern. To ensure optimality, $h(n)$ should be admissible (never overestimate). However, for practical speed, we often use a "weighted" A* where $h(n)$ is slightly aggressive to guide the search toward promising branches.

Component 1: Rotational Entropy (h_{rot})

We look at the set of remaining folded bends. We group them by their normal vectors.

Let N_{groups} be the number of distinct bend orientations remaining.

$$h_{rot}(n) = (N_{groups} - 1) \times \text{Cost}(90^\circ \text{ Rotation})$$

This assumes we can clear an entire side without leaving and returning. If we leave a side unfinished, $g(n)$ increases, and $h(n)$ remains high for the remaining bends, penalizing the branch.

Component 2: ABA Diversity (h_{aba})

We analyze the lengths of the remaining bends.

Let L_{groups} be the number of distinct bend lengths.

$$h_{aba}(n) = (L_{groups} - 1) \times \text{Cost}(ABA \text{ Change}) \times \alpha$$

Where α is a masking probability factor (e.g., 0.3), acknowledging that some of these changes might be hidden during rotations.

Total Heuristic:

$$h(n) = h_{rot}(n) + h_{aba}(n)$$

---

6. Proposed Implementation: "Optimal Bend Sequencing"

This section details the logic flow and provides the pseudo-code for the implementation. The architecture uses a SearchNode to represent the state and a PriorityQueue to manage the A* frontier.

6.1 Data Structures

To implement the algorithm effectively, robust data structures are required.

C++

```
// Enumeration for Action Types
enum ActionType {
    ACTION_BEND,
    ACTION_ROTATE,
    ACTION_ABA_RESIZE,
    ACTION_UNLOAD
};

// Machine Action Object for the Output Linked List
struct MachineAction {
    int id;
    ActionType type;
```

```
    double parameter;       // Angle (deg) or Length (mm)
    bool is_masked;         // True if this runs in parallel with another action
    double estimated_time;  // For cycle time calculation
    MachineAction* next;    // Pointer to next action
    MachineAction* prev;    // Pointer to previous action
};

// A* Search Node
struct SearchNode {
    // State Definition
    GraphTopology current_part_state; // Which bends are currently folded?
    double manipulator_angle;         // Current orientation (0-360)
    double aba_tool_length;           // Current width of the upper tool

    // Cost Metrics
    double g_cost;  // Cost from Start (Folded) to Here
    double h_cost;  // Estimated cost to Goal (Flat)
    double f_cost;  // g + h

    // Path Reconstruction
    SearchNode* parent;
    int causing_bend_id; // The bend that was "unbent" to reach this state

    // Comparison for Priority Queue
    bool operator>(const SearchNode& other) const {
        return f_cost > other.f_cost;
    }
};
```

6.2 The Main Solver Loop (Pseudo-Code)

The following pseudo-code implements the A* search in the **Reverse (Unfolding)** direction.

Python

```
FUNCTION FindOptimalSequence(BRepGraph FinalPart):
    # 1. Initialization
    TargetState = FlatPattern(FinalPart)
    StartState = FoldedPart(FinalPart)
```

```
# Priority Queue stores nodes, ordered by lowest f_cost
OpenList = PriorityQueue()
ClosedSet = HashSet() # To avoid cycles and redundant processing

# Create the root of the search tree
RootNode = CreateNode(StartState)
RootNode.g_cost = 0
RootNode.h_cost = CalculateHeuristic(StartState)
RootNode.manipulator_angle = 0 # Assume loading at 0 degrees
RootNode.aba_tool_length = MAX_LENGTH

OpenList.Push(RootNode)

# 2. Main A* Search Loop
WHILE OpenList is not Empty:
    CurrentNode = OpenList.PopLowestF()

    # Check for Success (Goal State Reached)
    IF CurrentNode.current_part_state == TargetState:
        RETURN ReconstructPath(CurrentNode)

    # Add to visited set
    Hash = ComputeHash(CurrentNode.current_part_state)
    IF ClosedSet.Contains(Hash):
        CONTINUE
    ClosedSet.Add(Hash)

    # 3. Expand State (Identify Valid Unbends)
    # We look for bends that exist in the current folded state
    # and check if they can be safely removed (flattened).
    CandidateBends = GetFeasibleUnbends(CurrentNode)

    FOR EACH Bend 'b' IN CandidateBends:

        # --- COST CALCULATION (The P4 Logic) ---

        # 4. Calculate Rotation Cost
        # Orientation of the bend relative to the machine
        TargetAngle = GetBendNormal(b)
        RotationDelta = Abs(TargetAngle - CurrentNode.manipulator_angle)

        Cost_Rotation = 0
        Time_Rotation = 0
```

```
        IF RotationDelta > 0.1:
            Cost_Rotation = HIGH_COST_ROTATION # e.g., 100
            Time_Rotation = RotationDelta / ROTATION_SPEED

        # 5. Calculate ABA Cost (Masked Time Logic)
        TargetLength = b.length
        LengthDelta = Abs(TargetLength - CurrentNode.aba_tool_length)

        Cost_ABA = 0
        Time_ABA = LengthDelta / ABA_SPEED

        # The Critical Check: Is ABA change hidden by Rotation?
        IF Time_ABA > Time_Rotation:
            # We pay only for the time exceeding the rotation
            Cost_ABA = MEDIUM_COST_ABA * (Time_ABA - Time_Rotation)
        ELSE:
            # Masked! No penalty.
            Cost_ABA = 0

        # 6. Create Successor Node
        New_G = CurrentNode.g_cost + Cost_Rotation + Cost_ABA + BASE_BEND_COST

        Successor = CreateNode()
        Successor.current_part_state = ApplyUnbend(CurrentNode.current_part_state, b)
        Successor.manipulator_angle = TargetAngle
        Successor.aba_tool_length = TargetLength
        Successor.parent = CurrentNode
        Successor.causing_bend_id = b.id
        Successor.g_cost = New_G
        Successor.h_cost = CalculateHeuristic(Successor.current_part_state)
        Successor.f_cost = Successor.g_cost + Successor.h_cost

        OpenList.Push(Successor)

    RETURN Failure("No feasible sequence found")
```

6.3 Helper Functions and Logic

6.3.1 GetFeasibleUnbends(Node)

This function acts as the gatekeeper, applying the geometric constraints defined in Section 4.

1. **Precedence Check:** Check the static Precedence Graph. Is this bend "locked" by another bend that is still folded? If yes, Skip.

2. **Collision Simulation:** Temporarily "flatten" the bend in the virtual B-Rep.
    - Does the flattening flange intersect the Manipulator?
    - Does the flattening flange intersect the Machine Throat?
    - Does the flattening flange intersect other parts of the sheet (Self-Intersection)?
3. **Manipulator Grip Check:** In this new state (with one more flat face), is there still a valid "Dead Zone" for the manipulator to grip? If the unbending operation consumes the last stable gripping face, this path is invalid (in the reverse logic).

6.3.2 ReconstructPath(Node)

The A* search produces a path from Folded to Flat. The machine needs Flat to Folded.

1. Trace Node.parent pointers from Goal back to Start.
2. Store the causing_bend_id in a list.
3. **Reverse the list.**
4. **Post-Processing:**
    - Iterate through the reversed list.
    - Insert explicit ROTATE actions where the angle changes between steps.
    - Insert explicit ABA_ADJUST actions where the length changes.
    - Mark ABA_ADJUST actions as masked = true if the preceding rotation is sufficient to cover the time.
5. Return the Head of the Linked List.

---

7. Commercial Benchmarking: Comparison with STREAMBEND

To validate the proposed model, we compare the derived logic against the documented capabilities of Salvagnini's OEM software, **STREAMBEND**.[15]

7.1 Automatic vs. Interactive Modes

The research confirms that STREAMBEND offers two primary modes [16]:

1. **Automatic Mode:** This corresponds directly to the **A*** algorithm described above. It runs the heuristic search to find a solution without user intervention.
2. **Interactive Mode:** This allows the user to manually override the priority queue. In our data structure, this would be equivalent to forcing the OpenList to pop a specific node chosen by the user, even if its f_cost is not the lowest. The GetFeasibleUnbends function would still run to prevent the user from selecting a physically impossible move (Collision), but the *optimality* is left to human judgment.

7.2 Simulation and Verification

STREAMBEND places heavy emphasis on "3D Virtual Simulation".[15] This validates our

requirement for a rigorous, hierarchical collision detection engine (Section 4.3). The simulation is not merely a visual aid; it is the visualization of the internal state transitions ($S_i \rightarrow S_{i+1}$) computed by the solver.

7.3 File Formats: P4 vs P4x

Snippet [16] notes that STREAMBEND generates two files:

- **P4 Program:** Contains the "Machine Language" – the bends, rotations, and ABA commands. This maps directly to our output MachineAction linked list.
- **P4x Program:** Contains shape information and 3D data. This maps to our BRepGraph and input geometry.
  This separation confirms that the "Sequencer" (which writes the P4 file) is distinct from the "Geometer" (which reads the P4x file), validating our modular software architecture.

---

8. Conclusion

This report has successfully reverse-engineered the core logic required for Phase 3 of the Salvagnini Panel Bender sequencing project. By analyzing the intersection of **Graph Theory**, **Kinematic constraints**, and **Heuristic Search**, we have derived a solution that mirrors the sophistication of industry-standard CAM systems.

Key Findings:

1. **The Dominance of Rotation:** The "Optimal" sequence is defined almost exclusively by minimizing manipulator rotations. The heuristic weights must reflect this disproportionate cost.
2. **The Power of Masked Time:** The P4's ABA system allows for "free" tool changes if synchronized with rotations. The cost function $g(n)$ must be non-linear to capture this "Shadow Cost," creating a strong algorithmic incentive to parallelize operations.
3. **Backward Planning Validity:** The constraints of the final 3D shape are far more restrictive than the flat sheet. Therefore, an **Unfolding Strategy** using A* is the most efficient path to a valid solution.
4. **Algorithmic Robustness:** The use of a Precedence Graph combined with Hierarchical Collision Detection ensures that the generated sequences are not just theoretical, but physically executable on the P4 hardware.

The proposed *A Solver**, driven by the **Weighted Rotation/ABA Heuristic**, provides a mathematically sound basis for the implementation of the FindOptimalBendSequence engine. This framework will allow the system to automatically generate collision-free, minimum-cycle-time programs for the Salvagnini P4, satisfying the project's Phase 3 objectives.

9. Appendix: Data Tables and Reference Models

9.1 Cost Function Weighting Matrix

The following table summarizes the recommended weights for the Cost Function $g(n)$, derived from the relative time impact of machine actions.

Machine Action	Relative Cost (W)	Time Basis (Approx.)	Notes
Bend Stroke	10	1.0s	Baseline unit.
Rotation (90^{c})	100	3.0s - 5.0s	Dominant factor. Includes acceleration/deceleration.
Rotation (180^{c})	120	4.0s - 6.0s	Marginal increase over 90 due to momentum.
Small Correction ($<$)	20	0.5s	fast alignment.
ABA Change (Masked)	0	0s	"Free" time hidden by rotation.
ABA Change (Unmasked)	$50 \times$	Variable	Linear penalty based on length delta.
Tool Setup (CLA)	500	30s+	Use of auxiliary blades (Manual/Auto). High penalty.

9.2 Linked List Node Structure (JSON Representation)

Example of the final output format for a simple 4-bend box.

JSON

Works cited

1. Range of automatic panel benders - SHM Consulting, accessed February 1, 2026, https://shmconsulting.hu/wp-content/uploads/2022/11/P4.pdf
2. The automatic hybrid adaptive Panel Bender. - Sanson Machinery Group, accessed February 1, 2026, http://www.sansonmachinery.com/wp-content/uploads/2013/12/P4Xe-UK-3962020312.pdf
3. Automatic determination of bending sequences for sheet metal parts with parallel bends, accessed February 1, 2026, https://www.researchgate.net/publication/233222153_Automatic_determination_of_bending_sequences_for_sheet_metal_parts_with_parallel_bends
4. Manufacturability-Driven Decomposition of Sheet Metal Products - Carnegie Mellon University Robotics Institute, accessed February 1, 2026, https://www.ri.cmu.edu/pub_files/pub1/wang_cheng_hua_1997_1/wang_cheng_hua_1997_1.pdf
5. p4-benders.pdf - Empire Machinery, accessed February 1, 2026, https://www.empire-machinery.com/wp-content/uploads/2021/06/p4-benders.pdf
6. Automatic Sheet Metal Bending Machine - Salvagnini P4, accessed February 1, 2026, https://www.salvagninigroup.com/en-US/products/panel-benders/P4
7. On sheet metal unfolding. Part 12: searching for feasible bends | by Analysis Situs | Medium, accessed February 1, 2026, https://analysis-situs.medium.com/on-sheet-metal-unfolding-part-12-searching-for-feasible-bends-2a91a8976431
8. Optimization of strip-layout using graph-theoretic methodology for stamping operations on progressive die - Semantic Scholar, accessed February 1, 2026, https://pdfs.semanticscholar.org/9ef0/898924d84ab09a47769578ac8b5447500f15.pdf
9. Collision detection during planning for sheet metal bending by bounding volume hierarchy approaches | Request PDF - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/publication/324742687_Collision_detection_during_planning_for_sheet_metal_bending_by_bounding_volume_hierarchy_approaches
10. Collision detection using boundary representation, BREP - Diva-Portal.org, accessed February 1, 2026, https://www.diva-portal.org/smash/get/diva2:808317/FULLTEXT01.pdf
11. Automatic Tool Selection in V-bending Processes by Using an Intelligent Collision

Detection Algorithm - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/publication/320419797_Automatic_Tool_Selection_in_V-bending_Processes_by_Using_an_Intelligent_Collision_Detection_Algorithm
12. Computer Aided Process Planning for Sheet Metal Cutting Operations in the Manufacturing Industry - Semantic Scholar, accessed February 1, 2026, https://pdfs.semanticscholar.org/e9dc/7732f377e7b1fa9dd29a920171f72a3611b7.pdf
13. Constraint-based process planning in sheet metal bending | Request PDF - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/publication/222082736_Constraint-based_process_planning_in_sheet_metal_bending
14. Planner for sheet metal components to obtain optimal bend sequence using a genetic algorithm | Request PDF - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/publication/220382093_Planner_for_sheet_metal_components_to_obtain_optimal_bend_sequence_using_a_genetic_algorithm
15. #3 Matt Humerick Salvagnini Software - YouTube, accessed February 1, 2026, https://www.youtube.com/watch?v=-gHDB6lRQ8A
16. Streambend en | PDF | Rotation | Art - Scribd, accessed February 1, 2026, https://www.scribd.com/document/761106869/Streambend-En
17. Sheet Metal Fabrication & Design Programming Software | STREAM, accessed February 1, 2026, https://www.salvagninigroup.com/en-US/products/software/STREAM