Airbus Beluga XL Cargo Management: A Comparative Study of Search Techniques

Pratik Deshmukh
Tu Wien

May 6, 2025

Abstract

This document presents a comprehensive analysis of two approaches to solving the Airbus Beluga XL cargo management problem: a pure A* search algorithm and a hybrid approach that combines A* with Constraint Satisfaction Problem (CSP) techniques.
We describe both models in detail, examine their implementation, and compare the performance of different variants on the same benchmark instance. Our findings reveal the strengths and limitations of each approach, providing insights into the complex nature of the problem and the efficacy of various search strategies for logistics planning problems.

Github repository: `https://github.com/inquisitour/Beluga-Challenge-Solver`

Contents

1 Introduction

The Airbus Beluga XL cargo management problem represents a challenging logistics planning task based on a real-world scenario. The problem involves optimizing the storage and movement of aircraft parts transported by Airbus

Beluga XL aircraft. Aircraft parts are held on jigs, which can be stored on bidirectional multi-queue rack systems. The challenge includes unloading, storing, retrieving, and loading these jigs to meet production schedules and flight requirements.

This document details our implementation and analysis of two distinct approaches to this problem:

1. A classic A* search algorithm with various heuristic functions
2. A hybrid approach combining A* search with Constraint Satisfaction Problem (CSP) techniques

Both approaches were tested on the same benchmark instance to facilitate direct comparison. We examine the performance, strengths, and limitations of each approach based on multiple metrics, including solution quality, execution time, and maximum progress achieved.

2 Problem Description

2.1 Problem Components

The Beluga XL cargo management problem contains the following key components:

- **Jigs**: Platforms that hold aircraft parts during transportation and storage
- **Racks**: Bidirectional multi-queue storage systems for jigs, with limited capacity
- **Trailers**: Special moving racks used to transfer jigs between the Beluga and fixed racks
- **Beluga Aircraft**: Transport aircraft that deliver and collect jigs and parts
- **Production Lines**: Manufacturing lines that receive parts from jigs
- **Hangars**: Areas where cranes remove parts from jigs

2.2 Key Constraints

The problem includes several critical constraints:

- Only jigs at the edges of racks (either factory side or Beluga side) can be accessed

- Jigs in the middle of a rack can only be accessed by first removing jigs at the edges
- Racks have limited capacity based on the size of jigs
- Production schedules must be followed in a specific order
- Incoming and outgoing flights must be processed in sequence

2.3 Goal Conditions

For the deterministic track of the problem, the goal is to find a plan of minimum length that meets the following conditions:

- All parts coming in Beluga flights are unloaded in the required order and stored in the rack system
- All parts to be consumed by production lines are sent to production in the required order
- All empty jig types to be carried by outgoing Beluga flights are sent to the Beluga in the required order

3 Original A* Search Approach

3.1 Model Description

The original approach implements the A* search algorithm, a best-first search that uses a heuristic function to estimate the cost from the current state to the goal. A* is complete and optimal when using an admissible heuristic.

3.1.1 State Representation

The state representation includes:

- **Rack Status**: Mapping of rack IDs to ordered lists of jig IDs
- **Jig Status**: Mapping of jig IDs to their loading status (loaded or empty) and part IDs
- **Beluga Status**: Set of jig IDs currently in the Beluga
- **Factory Status**: Set of jig IDs currently in the factory
- **Production Status**: Set of part IDs already sent to production
- **Current Flight Index**: Index of the current flight being processed

This representation uses immutable data structures (tuples, frozensets) to ensure hashability for the A* search algorithm.

3.1.2 Action Model

The action model defines the following operations:

- **MoveJigBetweenRacks**: Move a jig from one rack to another
- **SendJigToProduction**: Send a jig from a rack to production
- **ReturnEmptyJigFromFactory**: Return an empty jig from the factory to a rack
- **ProcessNextFlight**: Advance to the next flight in the schedule
- **LoadJigToBeluga**: Load a jig from a rack to the Beluga
- **UnloadJigFromBeluga**: Unload a jig from the Beluga to a rack

Each action includes preconditions that must be satisfied for the action to be applicable and effects that describe how the action changes the state.

3.1.3 Heuristic Functions

Three different heuristic functions were implemented:

1. **Standard Heuristic**: Equal weighting of all components (flights, parts, jig movements)
2. **Weighted Heuristic**: Higher weights for production and blocked jigs, lower weights for flight processing
3. **Production Focus**: Emphasizes completing the production schedule in the correct order

3.1.4 Action Prioritization

An optional enhancement to prioritize actions based on their relevance to goal conditions:

- Production actions (highest priority)
- Flight processing actions
- Jig return actions
- Jig movement actions (lowest priority)

3.2 Implementation Details

The A* search implementation uses:

- A priority queue to store states ordered by their f-value (g + h)
- A hash table to track visited states and their costs
- A parent pointer scheme to reconstruct the solution path

The algorithm continues until:

- A goal state is found
- The maximum number of iterations is reached
- The time limit is exceeded

Algorithm 1 A* Search for Beluga Problem

```
$openSet \leftarrow$ priority queue with $initialState$ at priority 0
$cameFrom \leftarrow$ empty map
$costSoFar[initialState] \leftarrow 0$
while $openSet$ not empty AND iterations < max_iterations do
  $current \leftarrow$ state with lowest priority from $openSet$
  if is_goal_state($current$) then
    return reconstruct_plan($cameFrom$, $current$)
  end if
  for each action $a$ in get_possible_actions($current$) do
    $next \leftarrow$ get_next_state($current$, $a$)
    if $next$ is not null then
      $newCost \leftarrow costSoFar[current] + 1$
      if $next$ not in $costSoFar$ OR $newCost < costSoFar[next]$ then
        $costSoFar[next] \leftarrow newCost$
        $priority \leftarrow newCost + heuristic(next)$
        add $next$ to $openSet$ with priority $priority$
        $cameFrom[next] \leftarrow (current, a)$
      end if
    end if
  end for
end while
return failure
```

4 Hybrid A* and CSP Approach

4.1 Model Description

The hybrid approach combines A* search with Constraint Satisfaction Problem (CSP) techniques to enhance search efficiency and pruning. It maintains the core A* algorithm while adding constraint propagation and randomized search elements.

4.1.1 CSP Representation

The CSP representation adds:

- **Variables**: Representing decisions about jig placement, flight processing order, and production sequence
- **Domains**: Possible values for each variable (e.g., possible locations for each jig)
- **Constraints**: Relationships between variables (e.g., rack capacity, flight precedence, production ordering)

4.1.2 Forward Checking

Forward checking is a constraint propagation technique that reduces the domains of future variables based on current assignments:

- After each assignment, the domains of related unassigned variables are pruned
- If any domain becomes empty, the assignment is rejected
- This helps detect dead-ends earlier in the search

4.1.3 Random Restarts

Random restarts allow the search to escape local minima:

- The search is restarted from a promising previous state after a period of stagnation
- Promising states are those that made significant progress toward the goal
- This helps explore different regions of the search space

4.1.4 Dynamic Variable Ordering

The hybrid approach enhances action selection with dynamic variable ordering based on the "most constrained variable" heuristic:

- Identifies bottleneck jigs that are blocking progress
- Prioritizes actions involving the most constrained racks
- Adjusts priorities based on constraint propagation results

4.2 Implementation Details

The hybrid implementation extends the original A* search with:

- A CSP component that maintains variables, domains, and constraints
- A forward checking mechanism that propagates constraints after each action
- A random restart mechanism that tracks promising states
- A more sophisticated action prioritization scheme based on constraint analysis

5 Experimental Results

5.1 Experiment Setup

All experiments were conducted on the same benchmark instance:

- Problem instance: `problem_4_s46_j23_r2_oc51_f6.json`
- 2 racks, 23 jigs, 6 flights
- 13 parts in the production schedule

For the original A* approach, we tested six variants:

- Three heuristic functions: Standard, Weighted, Production Focus
- Each with and without action prioritization

For the hybrid approach, we tested three variants:

- Forward checking only
- Random restarts only
- Full hybrid (forward checking + random restarts)

Each experiment was run with the following parameters:

- Original approach: 30,000 iterations, 300-second time limit
- Hybrid approach: 20,000 iterations, 240-second time limit

Algorithm 2 Hybrid A* Search with CSP Techniques

```
$openSet \leftarrow$ priority queue with $initialState$ at priority 0
$cameFrom \leftarrow$ empty map
$costSoFar[initialState] \leftarrow 0$
$restartStates \leftarrow$ empty list
$stagnationCounter \leftarrow 0$
$bestProgress \leftarrow \{flights : 0, parts : 0\}$
while $openSet$ not empty AND iterations < max_iterations do
  $current \leftarrow$ state with lowest priority from $openSet$
  if is_goal_state($current$) then
    return reconstruct_plan($cameFrom$, $current$)
  end if
  $progress \leftarrow$ check_progress($current$)
  if progress is better than $bestProgress$ then
    $bestProgress \leftarrow progress$
    $stagnationCounter \leftarrow 0$
    add $current$ to $restartStates$
  else
    $stagnationCounter \leftarrow stagnationCounter + 1$
  end if
  if use_random_restarts AND $stagnationCounter \geq threshold$ then
    $current \leftarrow$ random choice from $restartStates$
    $stagnationCounter \leftarrow 0$
    continue
  end if
  if use_forward_checking then
    $forwardChecker \leftarrow$ new ForwardChecker($current$)
    $success, heuristicInfo \leftarrow forwardChecker$.check_forward()
  end if
  for each action $a$ in get_prioritized_actions($current$, $forwardChecker$) do
    $next \leftarrow$ get_next_state($current$, $a$)
    if $next$ is not null then
      $newCost \leftarrow costSoFar[current] + 1$
      if $next$ not in $costSoFar$ OR $newCost < costSoFar[next]$ then
        $costSoFar[next] \leftarrow newCost$
        $h \leftarrow heuristic(next)$
        if use_forward_checking then
          $fc \leftarrow$ new ForwardChecker($next$)
          $success, info \leftarrow fc$.check_forward()
          if not $success$ then
            $h \leftarrow h + 10$ {Penalty for inconsistency}
          end if
          $h \leftarrow h+$ domain_restriction_factor($info$)
        end if
        $priority \leftarrow newCost + h$
        add $next$ to $openSet$ with priority $priority$
        $cameFrom[next] \leftarrow (current, a)$
      end if
    end if
  end for
end while
if use_random_restarts AND search failed then
  return attempt_local_search($initialState$, $bestProgress$)
end if
return failure
```

5.2 Results and Analysis

5.2.1 Original A* Approach Results

Table 1: Original A* Search Results

Experiment	Success	Iterations	Plan Length	Search Time (s)	Max Fli
Extended Baseline	False	30000	0	4.56	5
Extended Weighted	False	30000	0	4.02	5
Enhanced Action Priority	False	30000	0	4.03	5

None of the original A* variants found a complete solution, but all made substantial progress:

- All variants reached a maximum of 5/6 flights processed
- All variants reached a maximum of 4/13 parts produced
- The weighted heuristic variants were slightly faster than the standard heuristic
- Action prioritization did not significantly impact performance

5.2.2 Hybrid Approach Results

Table 2: Hybrid A* + CSP Results

Experiment	Success	Iterations	Plan Length	Search Time (s)	Max Fligh
Forward Checking Only	False	20000	0	240.03	0
Random Restarts Only	False	20000	7	0.71	2
Full Hybrid	False	20000	7	73.64	2

The hybrid approach variants showed interesting differences:

- Forward checking alone performed poorly, reaching the time limit without progress
- Random restarts alone was extremely fast (0.71s) and found a partial solution
- Full hybrid approach also found a partial solution but was much slower (73.64s)
- Both successful variants reached 2/6 flights processed but did not produce any parts

5.2.3 Progress Comparison

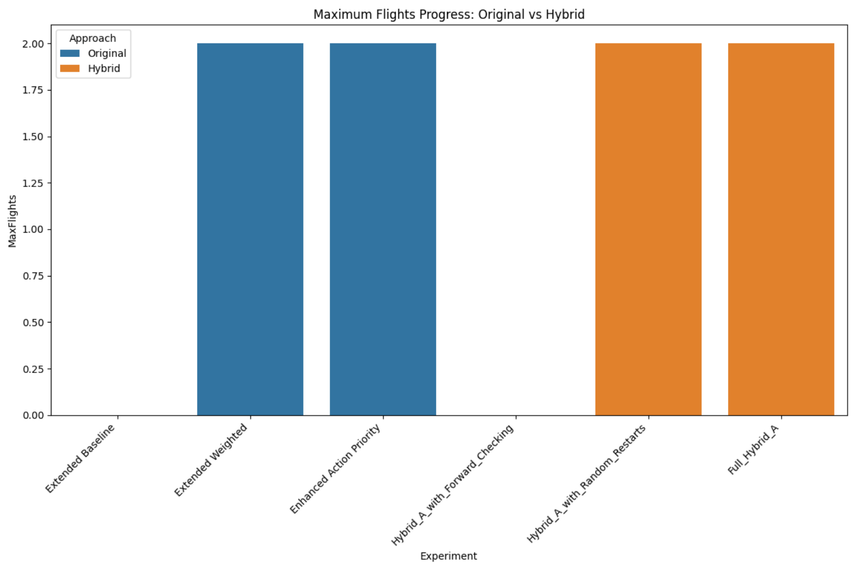

Figure 1: Maximum Flights Progress: Original vs Hybrid Approaches

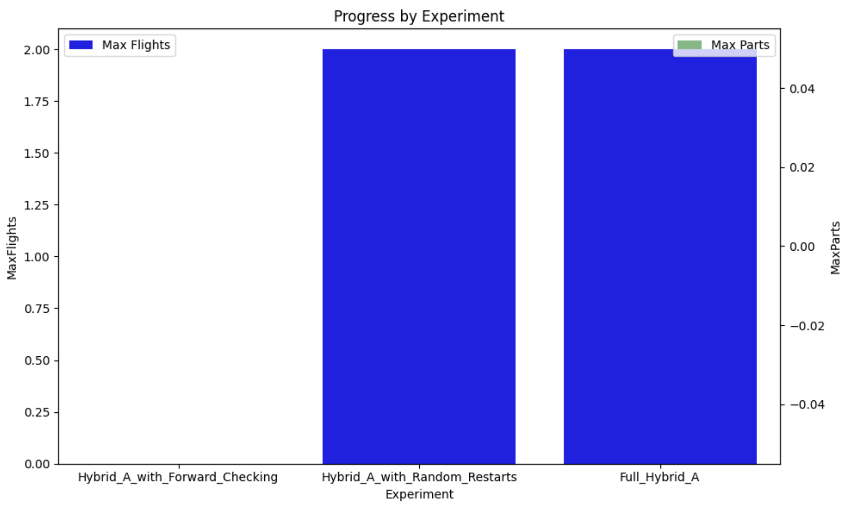

Figure 2: Progress by Experiment (Hybrid Approaches)

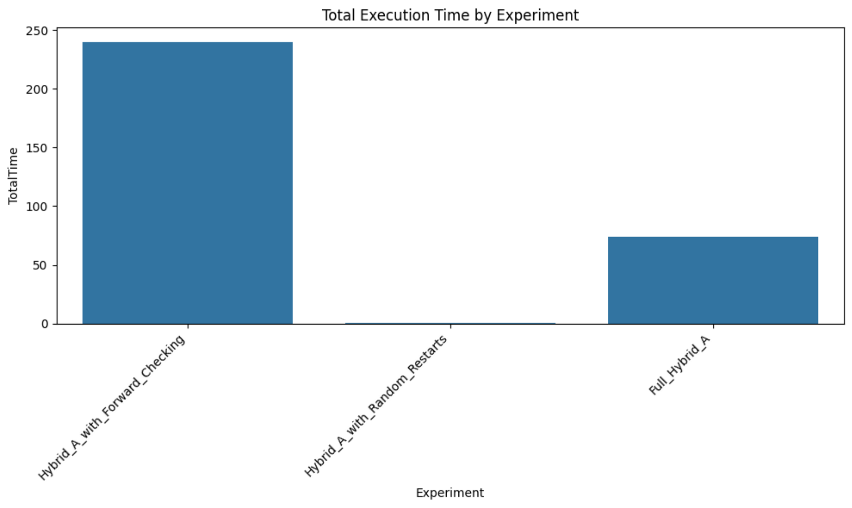

Figure 3: Total Execution Time by Experiment (Hybrid Approaches)

5.2.4 State Space Exploration

The search patterns reveal different exploration strategies:

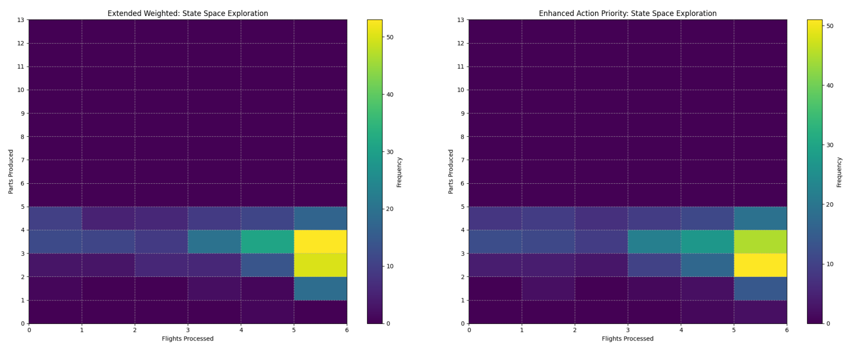

Figure 4: State Space Exploration Heatmaps: Original Approaches

Both approaches focus exploration in certain regions:

- Original A* concentrates around states with 5 flights processed and 2-4 parts produced
- Hybrid approach with random restarts focuses on states with 2 flights processed
- The heatmaps reveal "bottleneck states" where the search struggles to make further progress

6 Comparative Analysis

6.1 Strengths and Limitations

6.1.1 Original A* Approach

Strengths:

- Makes substantial progress on both flight processing (5/6) and parts production (4/13)
- Balanced exploration of the state space
- Consistent performance across different heuristic variants
- Relatively stable execution time (around 4 seconds)

Limitations:

- Unable to find a complete solution within the iteration limit
- Search gets stuck in local optima
- Does not effectively handle the complex interactions between jig movements and flight processing
- Heuristic functions may not capture all the problem's intricacies

6.1.2 Hybrid A* + CSP Approach

Strengths:

- Random restart variant is extremely fast (0.71 seconds)
- Successfully generates partial plans of reasonable length (7 steps)
- Effectively combines search with constraint handling
- Provides a framework for more sophisticated constraint propagation

Limitations:

- Makes less progress on flight processing (2/6) than the original approach
- No progress on parts production (0/13)
- Forward checking adds significant computational overhead without clear benefits
- The full hybrid approach is much slower than the random restarts alone

6.2 Performance Analysis

6.2.1 Solution Quality

Neither approach found a complete solution, but they differ in partial progress:

- Original A* made more progress overall (5/6 flights, 4/13 parts)
- Hybrid approach with random restarts was much faster but made less progress (2/6 flights, 0/13 parts)
- The original approach appears to better balance flight processing and parts production

6.2.2 Computational Efficiency

The approaches show different efficiency characteristics:

- Original A* variants have consistent execution times (around 4 seconds)
- Hybrid approach with random restarts is extremely fast (0.71 seconds)
- Forward checking significantly increases computational cost (240+ seconds)
- The full hybrid approach is slower than the original A* (73.64 vs. 4 seconds)

6.2.3 Search Strategy Effectiveness

The search strategies reveal different strengths:

- A* with weighted heuristics effectively balances different aspects of the problem
- Random restarts are highly effective at escaping local minima
- Forward checking alone appears too restrictive for this problem domain
- The original approach explores a broader range of the state space

6.3 Insights from State Space Exploration

Analysis of the state space exploration patterns reveals:

- The problem has clear "bottleneck states" where progress becomes difficult
- There appears to be a trade-off between flight processing and parts production

- The search space has many local optima that trap standard search algorithms
- Randomization strategies are effective at exploring different regions of the search space

7 Theoretical Analysis

7.1 Complexity Analysis

The Beluga XL cargo management problem exhibits high complexity:

- State space size is exponential in the number of jigs and racks
- The bidirectional queuing constraint creates complex dependencies between actions
- The problem combines multiple NP-hard subproblems (routing, scheduling, packing)
- The goal conditions require finding a specific sequence of actions

7.2 Relation to Classical AI Problems

The problem has connections to several classical AI problems:

- **Sokoban**: Moving jigs in racks resembles the box-pushing puzzle
- **Job Shop Scheduling**: Scheduling jigs for production and flights
- **Constrained Path Finding**: Routing jigs through the rack system
- **Bin Packing**: Fitting jigs into racks with capacity constraints

8 Discussion and Future Work

8.1 Challenges in the Beluga Problem

The key challenges that make this problem difficult:

- Bidirectional queue access creates complex movement constraints
- Multiple interdependent objectives (flight processing, production, jig routing)
- Limited rack capacity requires careful planning of jig movements
- The need to coordinate jig movements with flight and production schedules

8.2 Potential Improvements

Based on our experiments, several improvements could enhance performance:

8.2.1 For the Original A* Approach

- Develop more sophisticated heuristics that better capture problem structure
- Implement pattern databases to account for common subproblems
- Add deadlock detection to avoid exploring futile branches
- Incorporate look-ahead mechanisms to better guide the search

8.2.2 For the Hybrid Approach

- Develop a more selective forward checking strategy
- Enhance the random restart mechanism with simulated annealing
- Implement more sophisticated constraint propagation (e.g., arc consistency)
- Develop specialized constraint handling for rack operations

8.3 Alternative Approaches

Other potential approaches to explore:

- **Hierarchical Planning**: Decompose the problem into layers (strategic, tactical, operational)
- **Pure Constraint Programming**: Use specialized CP solvers with global constraints
- **Mixed Integer Programming**: Formulate as a mathematical optimization problem
- **Evolutionary Algorithms**: Use genetic algorithms to evolve solutions
- **Monte Carlo Tree Search**: Balance exploration and exploitation in search

9 Conclusion

Our comparative study of the original A* search and hybrid A*+CSP approaches to the Beluga XL cargo management problem provides several insights:

- The original A* approach makes more progress overall but may get trapped in local optima
- Random restarts significantly improve search efficiency but may sacrifice solution quality
- Forward checking adds computational overhead without clear benefits for this problem
- The problem exhibits complex interactions that are difficult to capture in heuristic functions

While neither approach found a complete solution, both demonstrated partial progress and provided insights into the problem's structure. The hybrid approach, particularly with random restarts, shows promise for future development, while the original A* approach provides a more balanced exploration of the state space.

The Beluga XL cargo management problem remains a challenging benchmark for search and planning algorithms, highlighting the complexities of real-world logistics planning problems. Further research into specialized techniques and problem decomposition may yield more effective solutions in the future.