Hybrid matheuristic for industrial-scale integrated production and transportation planning optimization

Marcin Kaminski[*] and Konstantin Kutzkov

4colors Research Ltd
Cambridge, UK

Garona is an integrated production and transportation planning solver developed to optimize the Airbus industrial system for the Airbus-BMW Group Quantum-Powered Logistic Challenge. Its main component is a hybrid matheuristic that combines mixed-integer linear programming with heuristics, quantum search techniques, and machine learning. Specifically, we introduce and employ two quantum analogs of classical optimization methods: quantum local search and quantum path relinking. The former is used to intensify the search, while the latter is employed to diversify it. Both components are powered by the same quantum routine, implemented in two ways: one based on amplitude amplification and the other on the Quantum Approximate Optimization Algorithm. In this report, we present the architecture, discuss the implementation and experiments, and share our findings and strategies for deployment.

I. INTRODUCTION

We begin by introducing the team, the company, and outlining our motivation for participating in the challenge.

Marcin Kaminski is a computer scientist with a background in both academic research and industry applications. He holds a master's degree in quantum computing from the Technical University of Gdansk, a PhD in operations research from Rutgers University, and a habilitation in theoretical computer science from the University of Warsaw.

Marcin specializes in discrete optimization and has published over 70 academic papers on topics such as combinatorial optimization, discrete algorithms, and graph theory. He led a research group at the University of Warsaw focused on graph algorithms and combinatorial optimization. In the industry, Marcin has worked on optimization projects with organizations such as AstraZeneca. He founded 4colors Research and has led multiple R&D projects, collaborating with various clients to address industrial-scale algorithmic challenges in optimization. His work effectively blends theoretical knowledge with practical applications.

Konstantin Kutzkov is a data scientist specializing in machine learning. He earned a master's degree in computer science from the University of Munich and a PhD from the IT University of Copenhagen. His research focused on machine learning on graphs and networks. He authored multiple research papers on machine learning and holds several industrial patents in this area.

Konstantin has developed his expertise through roles at renowned organizations, including the Bosch Center for Artificial Intelligence, NEC Laboratories, and Yahoo Labs. His diverse experience in these roles has equipped him with a deep understanding of machine learning techniques and their practical applications across various industries.

Founded in 2016 and based in Cambridge, UK, 4colors Research Ltd focuses on leveraging mathematical and computational techniques to help businesses solve complex problems. The company has successfully completed numerous projects, including developing efficient heuristics for combinatorial optimization, advancing data compression methods, and designing data-driven algorithms for complex network analysis. Collaborations with organizations such as Huawei Technologies and the UK's Ministry of Defence highlight 4colors' experience in providing practical algorithmic, optimization, and artificial intelligence solutions.

We chose to participate in the Airbus-BMW Group Quantum Computing Challenge because it perfectly aligns with our mission to apply advanced mathematical and computational methods to real-world challenges. We also find it a great opportunity to showcase how quantum methods can complement and enhance classical optimization frameworks.

II. SUBMISSION SUMMARY

Garona is an integrated production and transportation planning solver developed for this competition and designed specifically for the Airbus industrial system. This hybrid solver utilizes both classical and quantum computing techniques; however, the quantum components can be replaced with classical routines, allowing the solver to run entirely on classical resources. Conceptually, Garona's architecture can be viewed as a matheuristic, blending two complementary approaches: mathematical optimization and heuristics. The mathematical optimization methods, specifically mixed-integer programming, help identify promising regions of the solution space. Heuristics are then applied to search within these regions for better solutions and to improve existing ones.

Optimizing the Airbus industrial system, with its complexity, presents significant challenges. However, our model incorporates all the critical components and requirements outlined by Airbus, including designated production sites for each part, double sourcing strategies, production splits, fixed production targets for Final Assembly Lines (FALs), warehouses, transportation resources tailored to each part and route, as well as minimum, maximum, and target workload limits per site and supplier.

Given the project's limited timeline, we had to prioritize and simplify certain aspects. Specifically, we do not account for the temporal aspect of production and transportation, meaning that lead times are not optimized. Additionally, we apply simplified batching assumptions by calculating the volume of each part and assuming transportation resources can carry parts up to their maximum volume capacity.

We optimize simultaneously for three objectives: production cost, transportation cost, and CO2 emissions. These factors are integrated into a linear objective function with adjustable coefficients, allowing users to modify parameters and observe the effects of re-optimization. Additionally, target workload per site and supplier are treated as soft

[*]mjk@4colors-research.com

constraints, incorporated into the objective function with penalty coefficients.

We observed that the mixed-integer programming (MIP) model resulting from our formulation presents significant challenges for MIP solvers. Solvers often run for hours without finding any feasible solutions. Certain configurations, such as FAL production ratios or parameters like minimum/maximum/target site or supplier workload, can drastically increase the problem's complexity. However, linear programming (LP) relaxations can often be solved within minutes, or even seconds. Additionally, across different parameter sets, we consistently found that the number of non-integral variables in the LP relaxation solutions remains relatively small, only 10% to 20% of decision variables.

Garona leverages an external LP solver to compute an optimal solution to the LP relaxation of the problem. Once this solution is obtained, the next step is to round the non-integral variables to integer values, while the integral variables are kept fixed. Given that the number of non-integral variables is often too large to address the rounding optimally, these variables are processed in blocks. Importantly, this is a challenging optimization problem, as the non-integral variables often represent the most complex aspects of the problem — the ones that the external solver could not initially assign integer values to.

The block-based rounding strategy used by Garona can be viewed as a form of block coordinate descent, where the solver iteratively fixes groups of variables while striving to maintain both feasibility and integrality. The order in which these non-integral variables are processed is crucial to the overall solution quality. To address this, we employ reinforcement learning techniques that guide the selection process for which variables to round at each step. Finding the optimal rounding for a given set of variables is a complex optimization challenge.

To solve it, we introduce **Quantum Local Search** (QLS) and investigate two distinct methods: one inspired by Grover's algorithm, which utilizes amplitude amplification, and the other based on the Quantum Approximate Optimization Algorithm (QAOA). These quantum methods are compared to a classical approach that employs exhaustive enumeration for rounding block variables around a non-integral point. During the rounding process, we consider not only the objective function but also the feasibility of the constraints associated with the block variables. This consideration is crucial because the problem is non-linear, and represents a significant optimization challenge. Our findings indicate that both quantum methods perform well in practice and the quality of solutions is comparable with the classical exhaustive search.

QLS appears to be a novel application of quantum algorithms within the optimization framework. Surprisingly, to the best of our knowledge, this approach has not been addressed in the existing literature. The Grover-like methodology is especially noteworthy, as it provides a provable quadratic asymptotic speed-up. As quantum devices continue to scale and the size of the variable blocks increases, we anticipate a threshold where QLS will offer significant advantages over classical methods, primarily due to the benefits of quantum superposition.

The process of rounding the solutions may lead to infeasible outcomes, meaning we can obtain integral solutions that are not feasible. To address this, we periodically run a repair algorithm to restore feasibility. We utilize the *feasibility pump* [4], which is a linear program that has the same constraints as the LP relaxation of the MIP formulation but with an objective function designed to find the closest feasible solution to our integral rounded one.

The goal of the whole process is to identify integral feasible solutions, although on the way we encounter transitions between feasible non-integral solutions and infeasible integral points. For feasible non-integral solutions, we restore their integrality through rounding (i.e., using QLS), while for non-feasible integral solutions, we address their infeasibility with the feasibility pump. The feasibility pump has proven very effective in practice [2], and we have found it to work well for optimizing the Airbus industrial system as well.

Successful optimization algorithms must strike a balance between exploration and exploitation. They need to identify new promising regions of the solution space while also intensifying the search within a select few of these regions. In the approach outlined above, the LP solution highlights a promising area, and we utilize the QLS and feasibility pump to conduct a focused search within this space. While this method effectively helps us identify feasible integral solutions, we also require greater diversification to uncover other regions of the solution space where optimal solutions may be located. Expanding our exploration beyond the immediate vicinity of the initial LP solution will enhance our ability to discover more optimal solutions across the broader solution landscape.

We gather all the intermediate solutions generated during the process—both feasible non-integral and infeasible integral solutions—and create a population of elite solutions, selecting only the best among them. To enhance our optimization, we employ *path relinking* [6] on pairs of solutions within the elite set.

Path relinking is a classical optimization technique that operates on pairs of solutions, designating one as the base and the other as the guiding solution. We start with a base solution and, in successive steps, fix the variables where the two solutions differ one by one, generating new candidate solutions. The aim is that by transforming one good solution into another, we can uncover new solutions that are either better or that lead us to improved solutions compared to the originals. This iterative process helps us explore the solution space more effectively, potentially revealing optimal configurations that were previously overlooked.

We implement path relinking using the same quantum methods or classical exhaustive search as in our rounding process. The problem is tackled in blocks of variables, where for each block, we determine which variables should remain as in the base solution and which should be adjusted to match the solution. This formulation closely mirrors the rounding procedure, leveraging the same components. The result is **Quantum Path Relinking**, a novel approach that, to the best of our knowledge, has not been explored previously.

Our approach to tackling the Airbus-BMW challenge starts with classical optimization. We do not expect quantum systems, at least in the foreseeable future, to handle industrial-scale optimization problems as efficiently or independently as classical methods can. Instead, we believe the best path forward is to first understand the combinatorial nature of the problem and design hybrid algorithmic solutions around that. This has been our guiding principle in the development of Garona. Quantum methods are simply another set of tools in the optimization toolbox. Grover-like algorithms correspond to exact optimization methods (though probabilistic in nature), while techniques like QAOA can be viewed as hardware-based heuristics. The core challenge lies in optimization itself, whether tackled through classical or quantum means.

Incorporating new tools and finding synergies between classical and quantum approaches is crucial for large-scale industrial challenges, such as optimizing the supply chains of companies like Airbus or BMW Group. Even minor improvements in the solution or objective function can result in significant financial savings and positive environmental impacts.

On the quantum side, our emphasis has been on developing advanced algorithms, and we present experimental evidence of their performance on both simulators and quantum devices. While we recognize the significance of hardware-specific optimizations, particularly in the NISQ era, we have prioritized algorithmic advancements, which we believe are often overlooked in the quantum optimization practice.[1] We have identified several areas where quantum computing—especially techniques suited for NISQ devices—could offer significant advantages. Meanwhile, Garona can operate entirely on classical computing resources while also seamlessly integrating quantum capabilities, which will further enhance performance in the future.

III. DETAILED EXPLANATION

In this section, we provide a comprehensive overview of the algorithm, exploring both theoretical foundations and implementation specifics.

In Phase 1, our submission used dynamic programming, which was feasible due to the acyclic nature of the Product Breakdown Structure (PBS), allowing for a polynomial-time solution. However, during Phase 2, we recognized that the actual industrial problem is significantly more complex and necessitates a different approach. We conceptualize the problem by grouping production sites into clusters—these may consist of singletons (individual sites), suppliers, or even broader categories such as countries or regions. Each group can then be subject to knapsack-like constraints.

The objective is to optimize the logistics of transporting parts between various sites. Importantly, we can reduce this problem to the Travelling Salesman Problem (TSP), which highlights its NP-hardness when the Product Breakdown Structure (PBS) is included in the input.[2]

We have outlined most of our assumptions in the previous section. It is important to emphasize that this is a supply chain design optimization problem, which needs to be solved occasionally. As a result, we anticipate run times of several hours or even days, since the primary goal is to obtain the best possible solution, are allowed. Users are often willing to invest additional time if it leads to a better outcome.

However, we also recognize that the system may need to be run periodically to reoptimize in response to disruptions in production or transportation networks. While rapid result delivery is not a requirement, we believe it would be advantageous for planners to conduct experiments with different settings to understand how changes in input parameters affect the solutions. To facilitate this, the system should allow for quick turnaround times for these experiments.

A. Mathematical model

In this section, we present our formulation of the optimization problem as a mixed-integer program (MIP). (The model is implemented in `model.py`.) The problem can be conceptually divided into two primary components: production planning and transportation optimization. Production planning focuses on assigning parts to production sites while respecting hard constraints—such as double sourcing—and soft constraints, including workload targets. Transportation optimization involves selecting the most effective transport options.

A natural two-stage approach might involve first optimizing production based on a proxy for transportation costs (such as distance), followed by the selection of specific transportation options. While this method is certainly valid, it can lead to suboptimal solutions, as the two components are inherently interconnected; decisions made in one area directly influence the other. Consequently, we have chosen an integrated approach that incorporates both production and transportation into a single model, optimizing them simultaneously. In our initial experiments, we found that although this integrated approach presents a more complex problem, it remains tractable, and the results obtained are likely to be of higher quality.

Variables. There are 4 basic families of variables:

- `x[i][s]` are binary variables indicating whether part s is manufactured at site i,
- `y[i][s]` are integer variables indicating how many units of part s is manufactured at site i,
- `c[s][i][j][m]` are integer variables indicating how many units of part s are shipped from site i to site j by transportation resource m,
- `T[i][j][m]` are integer variables indicating how many transportation resources m are serving the route between sites i and j.

The solver also uses auxiliary integer variables that are usually sums of the basic variables: `t[s][i][j]` indicating how many units of part s are shipped from site i to site j, `supplier_workshare[i]` and `supplier_workshare[k]` indicating how much workload is assigned to site i and supplier k, respectively. Each of workload variables also have its version with `_deviation_below` and `_deviation_above` suffixes that are used to introduce absolute value of deviations from the target to the (linear) objective function.

Constraints. There are 6 families of constraints:

- production sites – ensures parts are produced only at sites capable of manufacturing those parts,
- double sourcing – requires each part to be manufactured at two sites, with a production split above a threshold that can be set individually for each part (e.g., 20% to 80%).
- fixed-rate production – allows users to select the number of aircraft manufactured at each FAL, optimizing the remainder based on partial requirements,

[1] Historically, algorithmic improvements have driven more progress in classical optimization [3, 8] and other computational fields [13] than hardware advancements alone.

[2] To illustrate, consider a TSP instance defined by a graph with n vertices and associated distances on the edges. By using a single transportation option, setting the capacity of each site to one, allowing all sites to manufacture all parts, and ensuring that transport can carry only one part at a time, we can model the PBS as a directed path of n vertices. The optimal manufacturing sites for subsequent parts in the PBS will align with a TSP path in the original graph, proving NP-hardness. Double sourcing can be accommodated by representing two copies of the graph.

- production limits – includes minimum and maximum production limits per site and supplier,
- warehouses – integrates storage solutions into the planning process,
- transportation resources – accounts for transportation options available for each part and route.

Objective. The linear objective function, that we minimize, consists of 4 components: `production_cost`, `transport_cost`, `emission_cost`, and `penalties`. The contribution of each can be adjusted using a corresponding user-defined parameter suffixed with `_obj _function_weight`. The penalty is the sum of the `_deviation_below` and `_deviation_above` variables over all sites and suppliers with user-defined weights `site_workshare_target_lambda_penalty` and `supplier_workshare_target_lambda_penalty`.

To illustrate the size of the problem, we initially have 115,242 variables and 81,254 constraints, with 233,717 non-zero entries in the model. However, the external solver effectively reduces this complexity, resulting in 70,206 variables, 4,924 constraints, and 145,746 non-zero entries. Despite this reduction, the model remains relatively large. According to the MIPLIB 2017 benchmark standards [5], only 10% of the problems in the MIPLIB dataset have more variables than our reduced model, while only 50% have more constraints or non-zero entries. This underscores the challenges in solving this MIP formulation, emphasizing why it is particularly difficult to solve.

B. Relax-and-fix

Our matheuristic approach was outlined in the previous section. Here, we aim to provide additional details and place the routine in a broader context. Relax-and-fix is a general strategy for solving MIPs [11], particularly for finding feasible solutions. This approach relies on the assumption that the LP relaxation is typically easier to solve (the relax part), allowing certain variables to return as integral, which can then be fixed (the fix part). It is important to note that while the optimal solution to the MIP may theoretically not coincide with these chosen variables, in practice, this often occurs. Even when it does not, the optimal MIP solutions are usually close to the values of the fixed variables. It is, however, essential to recognize that fixing some variables makes this a heuristic approach.

Once the integral variables are fixed, we are left with a partially integral solution, but our goal is to achieve all variables as integral. There are various methods to accomplish this; however, one common approach is the feasibility pump, which was mentioned in the previous section. This method is often preferred due to its practical performance. The idea is to round the non-integral variables. This rounding may lead to infeasibility, so we restore feasibility by projecting the infeasible partially-integral solution back to the feasible polytope. This process may cause some of the previously fixed variables to become free again. Consequently, we repeat the procedure, alternating between feasible non-integral and non-feasible integral solutions. Surprisingly, this routine converges quickly in practice. After several iterations, the number of integral variables typically becomes manageable for MIP solvers, and when it drops below a certain threshold, it can be effectively solved by a MIP solver.

There are various rounding strategies to consider when optimizing our model. It is often beneficial to evaluate how rounding a given variable up or down impacts both the objective function and feasibility. In practice, we generally prefer rounding methods that improve the objective function value while ensuring that all constraints remain satisfied. If such rounding is not feasible, we prioritize options that enhance the objective function value while minimizing the number of constraint violations and the extent of those violations.

Ideally, we would analyze all non-integral variables and evaluate every possible rounding option (both up and down) to compute their effects on the objective function and constraint violations. However, the number of potential options grows exponentially, making this approach impractical, especially given the large number of non-integral variables in our MIP.

To address this, we adopt a block-based strategy for fixing variables. We select a sufficiently small block of variables to allow either for exhaustive evaluation of all rounding options (classically), or amplitude amplification or QAOA (quantumly). For each block, we identify the best rounding decision based on the criteria mentioned above before moving on to the next block. We continue this process until all variables are fixed or until we deviate significantly from feasibility. If any infeasibilities arise, we employ the feasibility pump technique to restore feasibility and then repeat the process.

C. Reinforcement Learning

The partitioning of variables into blocks and the selection of those blocks significantly impact both the quality of the solutions obtained and the algorithm's speed. Like many decisions in optimization algorithms, these choices can be optimized through learning [1, 7]. To enhance our approach, we employ machine learning techniques to identify the most effective partitioning strategies.

Designing a machine learning model. The objective is to design a machine learning algorithms that selects variables for rounding that in subsequent iterations of the algorithm are most likely to result in an overall reduction of the number of variables assigned non-integer values by the solver.

At a high level the algorithm performs the following steps:

1. Initialize the LP model with a perturbated input.
2. Iteratively run the LP model and select variables to round using an ML model that evaluates if a given selection yields an improvement in scores. This step returns again 4 variables for which to perform (quantum) local search.
3. At step $k+1$, create a binary label set to 1 if the number of non-integer variables is strictly less than at step k, thus the labels can be seen as rewards.
4. Update the features and labels for the model. Retrain the ML model every k iterations.
5. When the model becomes infeasible, perturbate the input and re-initialize the LP model but not the ML model. (No feasibility pump is used.)

Instance perturbation. We generate different input instances by randomly sampling different FAL distributions such that the total sum of the five assembly lines Tianjin, Toulouse, Hamburg, Mirabel and Mobile is 100.

Feature engineering. At each step we describe the current state by storing all features that have non-zero values and all features with non-integer values. Further, we count the number of T, t and c-type variables with non-zero and non-integer values. We also collect the four selected variables at each iteration as well as features about the variable type and if two of the variables appear together in a constraint. Further, we record the given FAL distribution and the number of the iteration as features.

Model training evaluation. We use a Gradient Boosting model for predicting the if a given selection of variables would result in a decrease in the number of non-integer variables. We execute the reinforcement learning model for 500 random samples for the FAL distribution which results in above 6,000 examples.

In Figure 1 we show the performance of the model evaluated on a test dataset. Even if an AUC score of 0.622 is not particularly strong, the model does much better than random guessing. We are however convinced that the model has the potential for improvement, in particular by using more advanced techniques for learning a representation of the state of the model.

In Figure 2 we show the most important features detected by the model. As we see, the aggregated features about the number of variables of different types and the selected FAL distribution are the most important ones.

Leveraging the model. The trained machine learning model is used to recommend variables for rounding for the provided FAL distribution. We generate ℓ random tuples and select the one for which the ML model predicts the highest score. In Figure 3 we show the minimum number of non-integer variables found when running the LP model for 40 iterations. The case $\ell = 1$ corresponds to random sampling where the ML model plays no role. Overall, we observe considerable gains using predictions from the model. However, random sampling is important since scoring a large number of tuples of 4 non-integer variables yields worse results. This observation was consistent also for other FAL distributions.

D. Quantum Local Search

For a fixed block of variables, each can independently be rounded up or down. This introduces an exponential growth in the number of possible configurations as the number of variables in the block increases. We conceptualize this as keeping all other variables constant, allowing only the block variables to vary, which effectively explores the neighborhood of a non-integral point where all neighbors reside on the integral grid. To tackle this search efficiently, we designed two quantum approaches.

Amplitude amplification. In our encoding scheme, each of the k variables in the block is represented by a qubit: $|0\rangle$ indicates the variable is rounded down, and $|1\rangle$ means it is rounded up. We initialize a uniform superposition over these k qubits, allowing all rounding combinations to be represented simultaneously. The linear objective function is encoded in the phase of an ancilla qubit. This phase encoding is implemented via an oracle that applies controlled rotations based on the objective function coefficients. Specifically, each of the k qubits introduces a rotation in the phase, proportional to the corresponding coefficient in the linear objective function. Thus, the phase accumulated in the ancilla qubit is a weighted sum of these coefficients, dependent on the states of the block qubits.

Similarly, we represent linear constraints involving the block variables using additional ancilla qubits—one for each constraint. The encoding process mirrors that of the objective function, with each constraint encoded in the phase of a corresponding ancilla qubit. However, constraints also include a constant term, which we add to the phase via an additional rotation applied at the end of each constraint encoding.

Coefficients are normalized such that the phase angle remains between 0 and $\pi/2$, with smaller values more likely to result in the qubit states being $|0\rangle$ and larger values being $|1\rangle$. We then apply amplitude amplification, which consists of phase flipping and diffusion operations on the quantum register, followed by a measurement.

The core idea behind this approach is to employ a Grover-like algorithm to amplify states with minimal values in the ancilla qubits, thereby encouraging rounding configurations that both optimize the objective function and minimize constraint violations. This procedure is implemented in the function `quantum_local_search` in `quantum.py`.

QAOA. We also explore an alternative approach using the Quantum Approximate Optimization Algorithm (QAOA). To ensure compatibility with the libraries we employ, all numerical coefficients are converted to integers. We utilize a converter from `qiskit_optimization` to transform the binary program resulting from the block variables and the corresponding constraints into a Quadratic Unconstrained Binary Optimization (QUBO) formulation, where constraints are incorporated into the objective function, and violations are penalized.

Following this, we construct a QAOA instance using routines from `qiskit_algorithms`. The problem is then solved using the `MinimumEigenOptimizer`, with the COBYLA optimization algorithm employed to find the optimal solution. We use QAOA of depth 1. This procedure is implemented in the function `qaoa_local_search` in `qaoa.py`.

E. Quantum Path Relinking

As discussed in the previous section, we gather solutions generated throughout our relax-and-fix routine to compile a list of the best solutions identified so far, referred to as *elite solutions* in the literature. This list includes both feasible non-integral solutions and infeasible integral solutions, which we consider together.

We employ *path relinking* on pairs of selected solutions, with the expectation that superior solutions may be found along the path connecting two already promising solutions. It is important to note that the solutions produced through this process may not be feasible, integral, or may lack both properties.

To address this, we apply the feasibility pump to each of these solutions, aiming to reduce the number of infeasible variables. This step ensures that a Mixed-Integer Programming (MIP) solver can effectively tackle the corresponding integer program.

Path relinking follows a routine similar to the local search process described earlier, focusing on the variables that differ between the base and guiding solutions. For each of these differing variables, we must decide whether to retain the value from the base solution or to change it to the value in the guiding solution.

Given that the number of differing variables can be substantial, it becomes impractical to evaluate all of them simultaneously due to the exponential growth in possible combinations. Therefore, we divide the variables into manageable blocks. This decision-making process can be modeled similarly to the rounding of variables in the local search method. Fortunately, we can reuse the same modules developed for both classical and quantum local search for the path relinking process.

In the classical approach, we perform an exhaustive search over all block variables. The quantum approach utilizes previously developed quantum modules. The two quantum functions, `quantum_local_search` and `qaoa_local_search`, are designed to accept a set of linear constraints and an objective function. We configure the appropriate instances for both functions, enabling their use in either context.

For the quantum local search, we leverage quantum superposition and employ an algorithm that amplifies states with smaller objective function values while minimizing constraint violations. In the case of QAOA, we translate the linear program into a QUBO instance, incorporating penalties for constraint violations.

IV. IMPLEMENTATION

A. Practical implementation

We provide a practical demonstration of our work in the form of code and a short video. You will find the code at https://github.com/4colors-research/airbus-bmw-phase-II and the video at https://youtu.be/i0C0myunGOI.

B. Software and hardware

Our system was implemented in Python, using version 3.11.6 for code execution and experimentation. Although Python may not offer the speed of compiled languages, we chose it primarily for its seamless integration with Amazon Braket, Qiskit, reinforcement learning libraries, and its ease of prototyping. Additionally, the majority of our optimization tasks rely on external solvers and libraries, minimizing the need for custom high-performance code.

For linear and mixed-integer programming, we employed the HiGHS solver, an open-source tool developed at the University of Edinburgh. HiGHS is currently recognized as the fastest open-source solver on the Mittelmann MIPLIB2017 benchmark [10]. We used version 1.7.2 of HiGHS for our implementation.

For quantum circuit design, we used Qiskit 1.2.4. Circuits created with Qiskit were transpiled to Amazon Braket and executed on either local simulators, hosted simulators, or QPUs. For local simulation, we used both Amazon Braket's native simulators and Qiskit Aer. Hosted simulations were conducted on AWS's SV1 simulator. For QPU experimentation, we tested multiple options but conducted primary experiments on IonQ's Aria 1, which offered good availability, short wait times, favorable outcomes for our algorithms, and error mitigation support through debiasing.

For local testing and running MIP experiments, we used a desktop computer equipped with an Intel® Core™ i7-7820X processor (8 cores) and 64 GiB of memory, running on Ubuntu 23.10.

In both the quantum local search and quantum path relinking approaches, we use blocks of 4 variables to ensure that the number of resulting qubits and the sizes of the quantum circuits remain manageable.

C. Experiments and benchmarking

In this section, we present the experiments conducted to compare the performance of quantum routines against classical exhaustive search methods.

Most of our development was done on local and cloud-hosted simulators, while our experiments were conducted on both simulators and actual quantum devices. We evaluated two quantum routines: one based on amplitude amplification and the other on QAOA. Since both quantum local search and quantum path relinking use the same quantum routine, we generated 100 instances by combining outcomes from Garona runs with various initial parameters (e.g., minimum, maximum, and target workload) and objective function penalty weights to provide a diverse set of instances. The coefficients of the constraints were normalized by dividing each constraint's coefficients and constant term by the sum of the absolute values within each constraint. Likewise, the objective function coefficients were normalized by dividing each by the sum of their absolute values.

Our primary question was how well our quantum routines would perform relative to the classical exhaustive enumeration algorithm, assessing their scalability and potential as effective replacements to enhance solver performance.

We first focused on amplitude amplification. We benchmarked this routine on the 100 instances and tested it using the Amazon Braket local simulator and IonQ's Aria 1 via Amazon Braket. For each case, we sought to compare the algorithm's performance against classical exhaustive search. Since this is a multi-criteria optimization problem, aiming to both minimize the objective function and any infeasibility, we measured three parameters: the objective function value, the number of violated constraints, and the maximum violation.

For each instance, we ranked solutions from the classical approach, the quantum simulator, and the quantum device. Our ranking criteria were as follows: we prioritize solutions with the smallest maximum constraint violation; in case of ties, we prefer solutions with the fewest violated constraints; if ties persist, we select the solution with the lowest objective function value. We used 10, 100, and 1000 shots for the simulator and 100 shots for the QPU. To assess the performance of our quantum routines, we identified the best quantum solution on each simulator or QPU and checked its ranking within the classical exhaustive search results. Figures 4 and 5 display the results for the simulator and QPU, respectively.

Each figure is organized in a grid format, with the number of variables on the x-axis and the number of constraints on the y-axis. In the simulator figure, results are shown for 10, 100, and 1000 shots. Each cell in the grid visualizes how frequently (averaged over all instances) the best quantum solutions were ranked 1st, 2nd, 3rd, 4th, or 5th in the classical ranking. Analyzing each figure independently, we observe that performance decreases as the number of variables increases. This is expected, as the size of the search space doubles with each additional variable. However, the performance only slightly declines: the best quantum solution is among the top 3 classical solutions in nearly 50% of cases and among the top 5 in around 60% of cases. Naturally, as the variable count grows and the search space expands, more solutions approach the optimal. The details are discussed in Appendix: Experiment 1.

To validate this, we compared the best quantum (simulator) and classical solutions by averaging the differences across the three parameters: objective function, number of violated constraints, and maximum constraint violation. We observed changes across all three categories, but the objective function differences between cases with 3 and 6 variables were relatively small. Although the number of constraint violations and maximum violation increased, this is expected since additional variables per constraint increase the likelihood of infeasibility. Nonetheless, the numerical results remained reasonable, as discussed in Appendix: Experiment 2.

Finally, we also benchmarked our QAOA approach on the same 100 instances using the Qiskit Aer simulator. Similar to our previous experiments, we compared the algorithm's performance against classical exhaustive search using the same ranking method. The results are promising, though this method generates very large quantum circuits due to the introduction of slack variables to handle constraints. As a result, we could only test up to 5 variables and 3 constraints.

The performance was strong: in most cases, the QAOA solutions achieved a rank of 1 almost 100% of the time. However, in one instance, the solution ranked consistently at rank 4. This discrepancy is likely due to QAOA repeatedly converging on the same suboptimal energy configuration and failing to identify the optimal solution. This outcome highlights a potential limitation of QAOA; while it produced excellent results, it may be less robust than amplitude amplification and may not scale as efficiently. Nonetheless, QAOA has shown high potential in smaller instances, even if its scalability remains uncertain.

V. CONCLUSION & OUTLOOK

We developed Garona, a hybrid solver for the integrated production and transportation problem. The model incorporates all essential business constraints outlined by Airbus, such as support for dual sourcing, indirect routing through warehouses, and the minimization of production, transport costs, and CO2 emissions. Its current limitations include the absence of temporal considerations and lead time minimization. Despite these limitations, the problem remains highly challenging for MIP solvers to solve exactly, motivating the need for innovative approaches.

To address this, we designed an optimization architecture that blends classical and quantum techniques within a matheuristic framework, combining mathematical optimization with heuristic elements. This approach is aimed at planners who prioritize solution quality over runtime, allowing for execution times on the order of hours if needed. The core emphasis is on locating integral solutions close to those of the relaxed model and refining them with additional methods. The primary contribution of the quantum modules is to explore solution spaces for small binary linear programs, minimizing the objective function and constraint violations simultaneously. Quantum features such as superposition, amplitude amplification, and QAOA play a central role in this process.

Our experimental results demonstrate the viability and scalability of this approach on modern NISQ QPUs, suggesting that these quantum components are promising for future problem sizes as quantum processing capabilities expand. To the best of our knowledge, the Quantum Local Search and Quantum Path Relinking methods are novel, represent original contributions to the field, and may be of independent interest. This hybrid framework illustrates the potential of integrating quantum algorithms into classical optimization to advance solution quality.

Garona is ready for practical application, offering solutions using either classical or hybrid classical-quantum resources. Below, we outline key areas for further development, deployment considerations, and improvements required to transition Garona to production.

A. Software development

The solver could be significantly improved by rewriting time-critical sections in a compiled language like C++ to enhance execution speed. Additionally, introducing multithreading or multiprocessing to handle multiple optimization tasks concurrently would also enhance performance. For example, the entire algorithm could be run with different seed values to increase the likelihood of finding an optimal solution, or individual heuristic steps could be executed in parallel. This would require further experimentation to determine the best parallelism strategies for accelerating the algorithm.

Another improvement involves using incremental techniques. Calculating the objective function is often computationally expensive, but since changes in the solution are typically local, the objective function can be updated incrementally based on these small adjustments rather than recalculating it in full each time. This technique, commonly applied in optimization, can significantly reduce computation time.

Finally, integrating an alternative external solver could significantly enhance Garona's performance. For instance, Gurobi was found to be approximately ten times faster than HiGHS, the solver we are currently using, during its last evaluation on the Mittelmann benchmark [10]. Transitioning to a commercial solver like Gurobi could therefore yield substantial speed improvements.

B. Model

Extensive effort was devoted to ensuring model accuracy; however, comprehensive validation by industry experts is still needed. This includes reviewing and confirming the quality of generated solutions and benchmarking them against baseline solutions produced by existing Airbus algorithms. Additionally, we would like to benchmark our algorithm on the BMW dataset and potentially adapt the model to meet BMW's specific requirements.

An obvious extension of the model would be to incorporate temporal factors into the optimization, such as lead times, and enable minimization for these and potentially other objectives as required.

The model can support triple sourcing (in fact, it can enforce any k-sourcing constraint, where k factories are active for each part). However, in testing, we found that triple-sourcing instances were infeasible, primarily due to limited connectivity in the transportation network and other constraints. It would be valuable to evaluate the solver's performance with an augmented dataset to see how it handles triple (or higher) sourcing constraints.

C. Optimization

Our matheuristic approach utilizes LP relaxation as a foundation for identifying high-quality solutions. Enhancing this framework with additional heuristics could significantly improve the search process by broadening solution strategies. For example, integrating metaheuristics such as genetic algorithms or other evolutionary techniques could offer greater diversity and robustness in solutions. This expanded approach could also incorporate various constraint-aware local search methods, including tailored rounding techniques that respect constraints for more precise refinements.

Expanding the set of repair methods, especially those that complement the feasibility pump, would further improve solver robustness. Additionally, introducing branching within the algorithm and extending our reinforcement learning-based variable selection approach to also prioritize branching decisions could boost efficiency. By training the model to select specific variables or blocks to branch on first, we can potentially achieve faster convergence than relying on branching in the external MIP solver.

An alternative approach could involve dynamically adjusting certain constraints. For instance, setting stricter minimum and maximum workload allocations for sites and suppliers, or applying higher penalties for deviations from target workloads, often results in more challenging problem instances. Instead, it may be beneficial to begin with parameters that yield easier instances. These solutions, even if they do not fully satisfy the original constraints, could then be used to warm start the optimization. As we iteratively adjust the constraint parameters, this staged approach can drive the solution process toward feasible and optimal solutions for the true problem instances we aim to solve.

D. Quantum

There are several ways to enhance the integration of quantum computing in our algorithm. First, we should focus on optimizing the quantum circuit sizes for both amplitude amplification and QAOA approaches. The size of each subproblem solved quantumly is determined by the number of variables and constraints, with one qubit allocated per variable and one auxiliary qubit per constraint. To minimize the use of ancillary qubits, we could group or combine constraints. While we have experimented with this optimization, it appears to affect the results, and further work is needed to optimize the number of qubits effectively.

In the case of QAOA, we use one qubit per variable, with all ancillary qubits generated by routines from `qiskit_algorithms`. A careful inspection and manual construction of the circuit may reduce the number of qubits and gates used, which can further optimize performance.

Another potential improvement involves exploring larger neighborhoods when more qubits are available. For rounding decisions, we currently use binary decisions (rounding up or down). However, we could consider larger neighborhoods by examining integral values within the variable domain that are close to the value being rounded. This approach is particularly relevant for variables that must be integral rather than binary. Our classical experiments have shown that increasing the search neighborhood to 4 significantly improves algorithm performance. However, this is only feasible if sufficient qubits are available, as larger neighborhoods require more qubits to encode specific decisions. For instance, a neighborhood of 4 (two below and two above the non-integral value) would necessitate 2 qubits per variable, effectively doubling the qubit requirement compared to the current implementation.

NISQ devices present several challenges, including issues with fidelity. There are likely gains to be made by optimizing our quantum routines. Currently, we rely on IonQ's error mitigation through debiasing, but several other techniques—such as zero-noise extrapolation, probabilistic error cancellation, or measurement error mitigation—may be beneficial as we scale up the number of variables and build larger circuits.

On the algorithmic side, there is significant room for improvement. Garona is an involved algorithm with numerous components that could benefit from incorporating quantum routines as alternatives to classical methods. One example is the exploration of quantum genetic algorithms. Although such algorithms have been discussed in the literature [9, 12] and may be too complex for NISQ devices, certain aspects can still be implemented. For instance, parallel computation of the objective function for multiple individuals in the population merits further investigation.

Quantum algorithms inherently incorporate randomness, and while quantum local search (QLS) is designed to align with local search techniques, QAOA introduces even more variability. Running these routines multiple times often yields different results. Similar to classical parallelization, we could extend a parallel approach to quantum solvers. Running multiple QAOA instances or quantum local searches with varying initial states, constraints, or QPU backends could enhance our exploration of the solution space, diversify potential solutions, and improve robustness.

As previously mentioned, QAOA currently involves transforming constraints into penalties within a QUBO formulation. Fine-tuning these penalty terms could enhance the feasibility and optimality of solutions derived from quantum computations. Investigating adaptive penalty schemes, where the penalty strength dynamically adjusts based on solution feasibility, may lead to faster convergence on integral and feasible solutions.

Digital annealing devices may also serve as a natural choice for implementing quantum modules. The same QUBO instances we produce for QAOA could be solved using devices like D-Wave. This approach would be applicable for both rounding and path relinking. The advantage of these systems is their potential to optimize larger blocks of variables than those currently manageable on gate-based QPUs. Although we have not yet experimented with this approach, our previous experiences suggest that they could efficiently handle block sizes of a significantly larger order of magnitude.

Acknowledgements

We would like to express our gratitude to the Airbus and BMW teams for their prompt responses to our inquiries and for sharing their expertise on the optimization problem and related data. We also extend our thanks to Amazon Web Services for their generous support through credits, as well as their invaluable technical assistance. Additionally, we greatly appreciate the efforts of the organizers and The Quantum Insider team for putting together this challenge, which has been a truly enjoyable experience for us.

[1] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[2] T. Berthold, A. Lodi, and D. Salvagnin. Ten years of feasibility pump, and counting. *EURO Journal on Computational Optimization*, 7(1):1–14, 2019.

[3] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations research*, 50(1):3–15, 2002.

[4] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.

[5] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, et al. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

[6] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

[7] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[8] T. Koch, T. Berthold, J. Pedersen, and C. Vanaret. Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10:100031, 2022.

[9] R. Lahoz-Beltra. Quantum genetic algorithms for computer scientists. *Computers*, 5(4):24, 2016.

[10] H. Mittelmann. The MIPLIB2017 benchmark instances. `https://plato.asu.edu/ftp/milp.html`, 2024.

[11] Y. Pochet and L. A. Wolsey. *Production planning by mixed integer programming*, volume 149. Springer, 2006.

[12] B. Rylander, T. Soule, J. A. Foster, and J. Alves-Foss. Quantum genetic algorithms. In *GECCO*, page 373, 2000.

[13] Y. Sherry and N. C. Thompson. How fast do algorithms improve? *Proceedings of the IEEE*, 109(11):1768–1777, 2021.

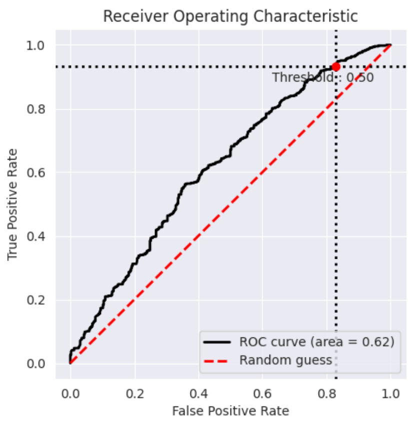

FIG. 1: The ROC curve for the test dataset.

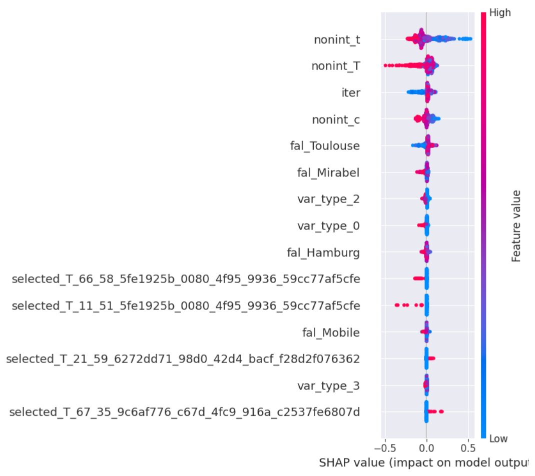

FIG. 2: The most important features according to their Shapley values.

Appendix: Experiment 1

In this experiment, we compare the best solution found by the quantum approach and assess its ranking among classical solutions. The quantum and classical solutions are ranked according to the following criteria: first, we prioritize solutions with the smallest maximum constraint violation; in the case of ties, we prefer solutions with the fewest violated constraints;

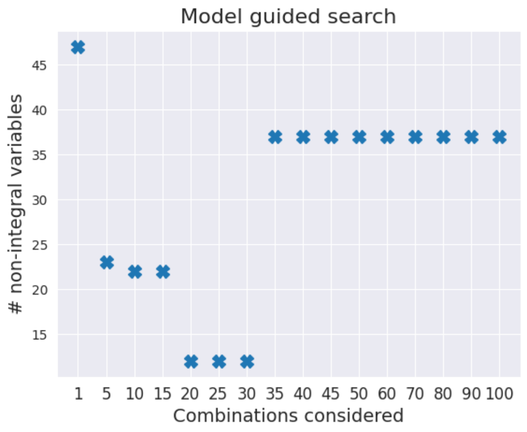

FIG. 3: Minimum number of non-integer variables found for a varying number of random tuples ℓ.

if ties persist, we choose the solution with the lowest objective function value. The benchmark consists of instances derived from the rounding and path-relinking components of Garona. The experiments were conducted over 100 instances. We used the Amazon Braket local simulator with 10, 100, and 1000 shots for the quantum routine. For the classical algorithm, we employed exhaustive search, while for the quantum approach, we used the amplitude amplification routine described in Section III D.

Figure 4 is arranged as a grid, with the number of variables on the x-axis varying between 2 and 6 and the number of constraints on the y-axis varying between 2 and 5. Each individual plot represents results from three experiments, one each with 10, 100, and 1000 shots. Each experiment was conducted over 100 instances, and we present the frequency with which the best quantum solution ranked as the 1st, 2nd, 3rd, 4th, or 5th compared to the classical solutions according to the ranking method described above.

Appendix: Experiment 2

In this experiment, we compare the best solutions obtained from classical and quantum approaches on benchmark instances derived from the rounding and path-relinking components of Garona. Tables I, II, III contain data on the differences in the objective function, the number of violated constraints, and the maximum constraint violation, respectively. The instances used involved 3 and 6 variables with between 3 and 6 constraints.

The experiments were conducted over 100 instances. We utilized the Amazon Braket local simulator with 1,000 shots for the quantum routine. For the classical algorithm, we employed exhaustive search, while for the quantum approach, we implemented the amplitude amplification routine described in Section III D. The best solution from each approach was selected based on the following criteria: first, we prioritized solutions with the smallest maximum constraint violation; in case of ties, we preferred solutions with the fewest violated constraints; and if ties persisted, we chose the solution with the lowest objective function value.

A negative value in a cell indicates that the classical solution was superior, while a positive value suggests that the quantum solution performed better. For example, when comparing instances with 4 constraints, we observe the differences between the 3-variable and 6-variable scenarios, effectively doubling the problem size. The quantum computer found instances for 3 variables with a slightly lower objective function value (-0.009450), whereas for 6 variables, the quantum algorithm identified instances with a higher objective function value (0.020027). This discrepancy arises from our multi-criteria selection process, as described earlier.

Next, we can examine the average number of violated constraints. The quantum approach is only slightly worse than the classical solution for 3 variables (-0.02), while for 6 variables, it violated more constraints (-0.38), which explains the observed gain in the objective function. Additionally, when comparing the maximum constraint violations, we note that the value was much lower for 3 variables (-0.014302) than for 6 variables (-0.200267). The improvement in the objective function is offset by small constraint violations, which may represent a beneficial trade-off since these minor violations can potentially be addressed in subsequent runs of the matheuristic.

It is important to note that the algorithm scales well with the number of variables and constraints. For instance, in the case of 6 constraints with 3 variables, the quantum solution satisfied more constraints than the classical solution (0.02). However, with 6 variables, only 5% more constraints were not satisfied by the quantum algorithm compared to the classical one. These are relatively small numbers, and it is likely that these infeasibilities can be rectified by the algorithm in later iterations. Moreover, the quantum approach offers a significant speed-up over classical exhaustive search methods.

Variables \ Constraints	3	4	5	6
3	0.023263	-0.009450	-0.019473	0.022789
6	-0.076352	0.020027	-0.004650	0.129572

TABLE I: The mean difference between the best quantum and the best classical solutions in the objective function.

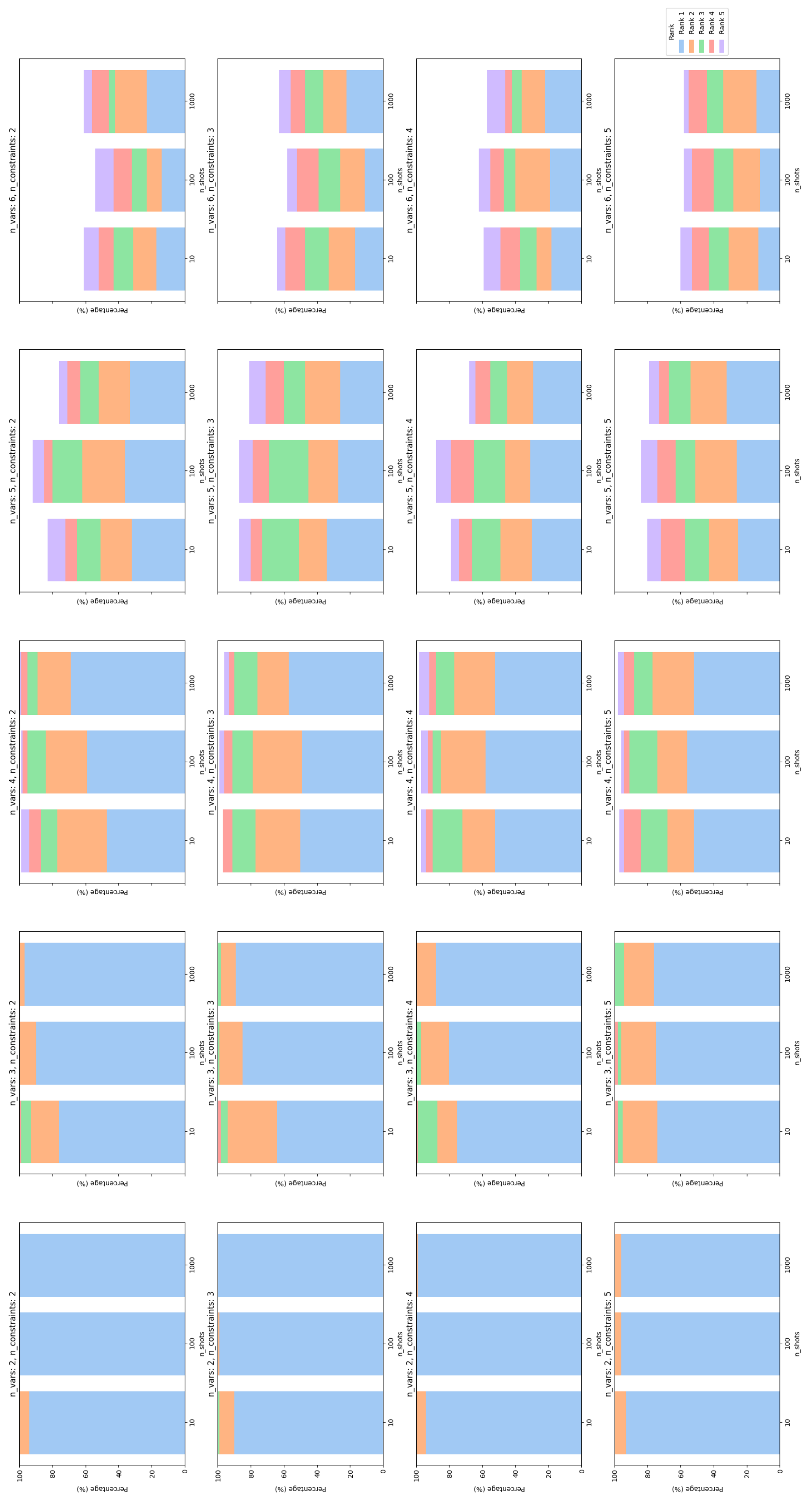

FIG. 4: The percentage of the best quantum solutions that are ranked 1st, 2nd, 3rd, 4th, or 5th in comparison to the classical solutions.

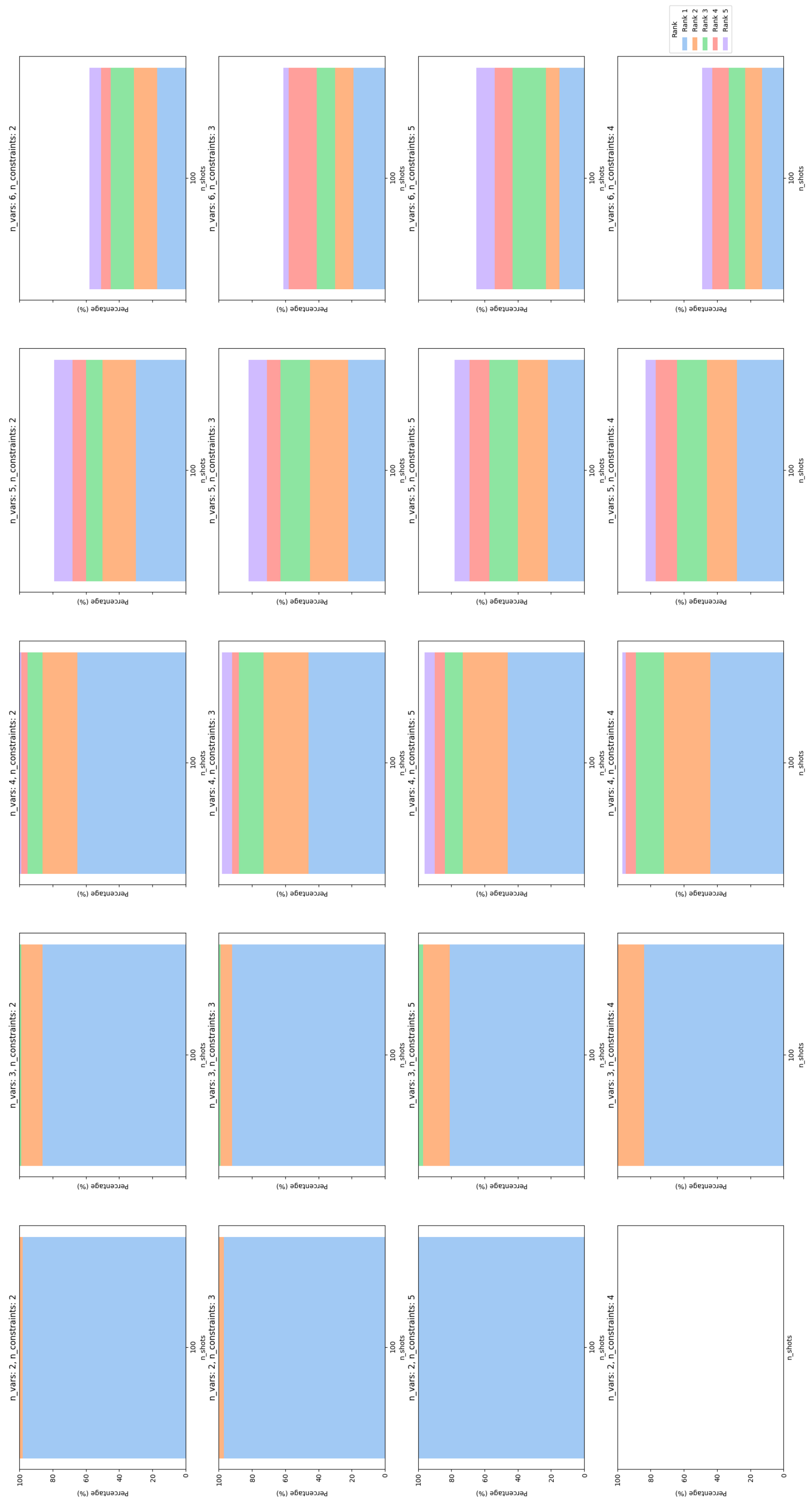

FIG. 5: The percentage of the best quantum solutions that are ranked 1st, 2nd, 3rd, 4th, or 5th in comparison to the classical solutions.

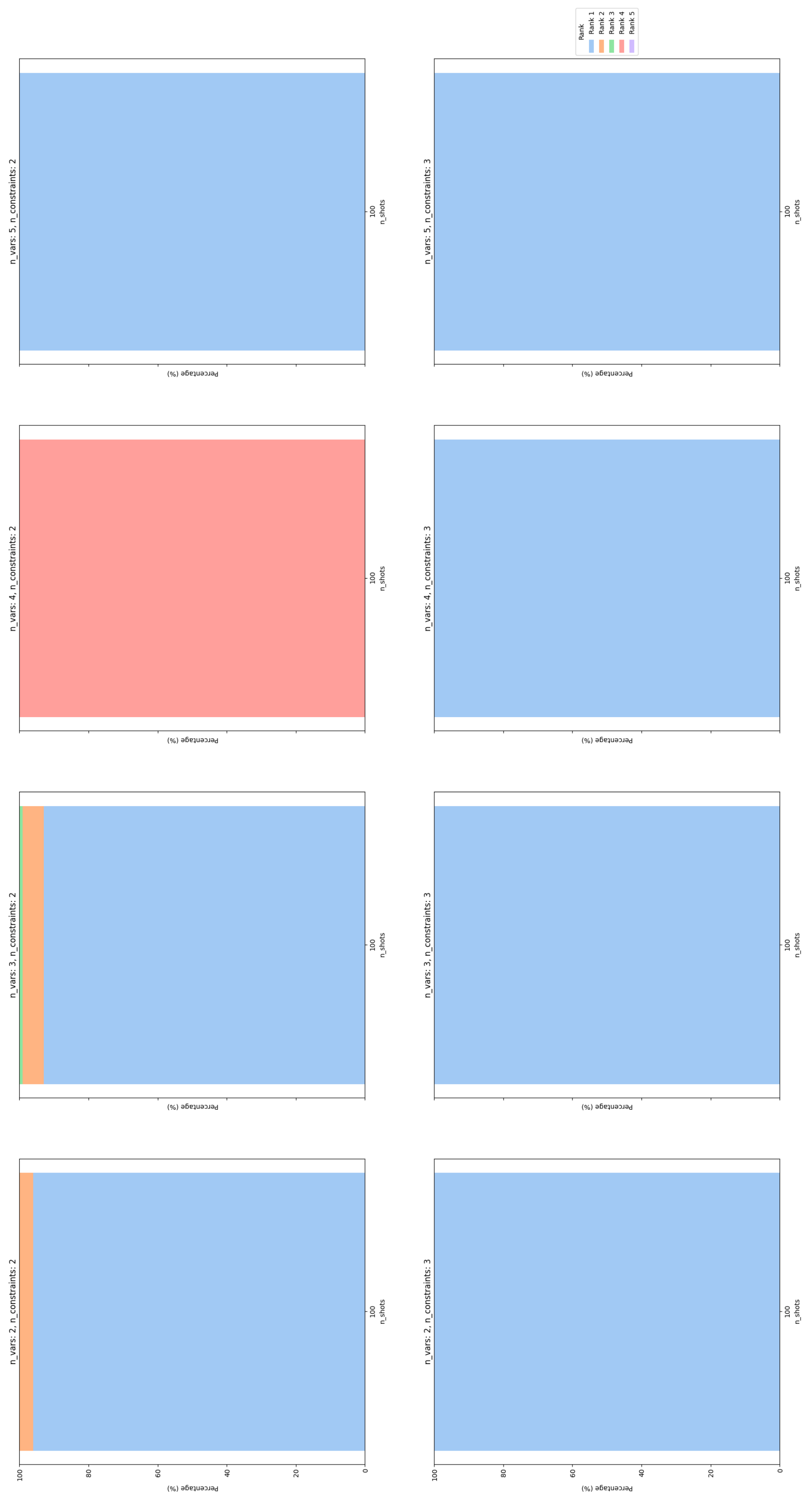

FIG. 6: The percentage of the best quantum QAOA solutions that are ranked 1st, 2nd, 3rd, 4th, or 5th in comparison to the classical solutions.

Variables \ Constraints	3	4	5	6
3	0.01	-0.02	0.00	0.02
6	-0.48	-0.38	-0.41	-0.34

TABLE II: The mean difference between the best quantum and the best classical solutions in the number of violated constraints.

Variables \ Constraints	3	4	5	6
3	-0.018149	-0.014302	-0.041896	-0.058143
6	-0.208464	-0.200267	-0.250934	-0.289694

TABLE III: The mean difference between the best quantum and the best classical solutions in the maximum constraint violation.

Variables \ Constraints	2	3	4	5	6
2	46 (16 | 6)	57 (18 | 7)	68 (20 | 8)	79 (22 | 9)	90 (24 | 10)
3	60 (17 | 8)	73 (19 | 9)	86 (21 | 10)	99 (23 | 11)	112 (25 | 12)
4	74 (18 | 10)	89 (20 | 11)	104 (22 | 12)	119 (24 | 13)	134 (26 | 14)
5	88 (19 | 12)	105 (21 | 13)	122 (23 | 14)	139 (25 | 15)	156 (27 | 16)

TABLE IV: The size of the quantum circuit in the amplitude amplification routine is given as a function of the number of variables and constraints. The format is: the number of gates (circuit depth | circuit width).

Appendix: Parameters

Each run of Garona builds the instance from the instance data in `data` directory but supplemented by the parameters in `instance-input.json` file. The following settings allow for tuning the problem-specific requirements and objective function priorities:

- `default_production_split`: defines the lower bound for production split between sites producing the same part. For example, setting this to 0.2 means that production is split between sites at a minimum of 20% and a maximum of 80%.

The next four parameters represent the coefficients in the objective function's linear combination, each controlling the contribution of a specific element to the overall optimization objective. You can interpret this function in terms of monetary value:

- `production_cost_obj_function_weight`
- `transport_cost_obj_function_weight`
- `emission_cost_obj_function_weight`
- `target_workshare_obj_function_weight`

For example, the `emission_cost_obj_function_weight` could represent the current price of CO2 certificates per gram. The `target_workshare_obj_function_weight` applies a penalty to deviations from workload targets, with its coefficient representing the penalty cost per euro that deviates from the target workload.

Additional workload-related parameters:

- `site_workshare_target_lambda_penalty`: controls the importance of meeting workload targets for production sites.
- `supplier_workshare_target_lambda_penalty`: controls the importance of workload targets for suppliers.

The `demand` parameter specifies the total number of aircraft that all Final Assembly Lines (FALs) must produce. `demand_per_FAL` allows for defining production targets for individual FALs, specifying how many aircraft each should produce. The sum of all `demand_per_FAL` entries should not exceed the demand value. If the total is less than demand and some FALs do not have specified values, the solver will automatically distribute the remaining production optimally across the unscheduled FALs.

For example, in the setup below, production is set as follows, Hamburg: 20, Mirabel: 30, Mobile: 20. The remaining production demand (30 units) will be optimally allocated across Tianjin and Toulouse.

```
demand_per_FAL : {
"Tianjin FAL" :  10,
"Toulouse FAL" :  30,
"Hamburg FAL" :  30,
"Mirabel FAL" :  10,
"Mobile FAL" :  10
}
```