From Multi-Agent Pathfinding to 3D Pipe Routing

Gleb Belov,1 Wenbo Du,2 Maria Garcia de la Banda,1 Daniel Harabor,1 Sven Koenig,3 Xinrui Wei1

1Monash University, 2Australian National University, 3University of Southern California
gleb.belov@monash.edu, u6361796@anu.edu.au, maria.garciadelabanda@monash.edu,
daniel.harabor@monash.edu, skoenig@usc.edu, xwei0012@student.monash.edu

Abstract
The 2D Multi-Agent Path Finding (MAPF) problem aims at finding collision-free paths for a number of agents, from a set of start locations to a set of goal locations in a known 2D environment. MAPF has been studied in theoretical computer science, robotics, and artificial intelligence over several decades, due to its importance for robot navigation. It is currently experiencing significant scientific progress due to its relevance for automated warehouses (such as those operated by Amazon) and other important application areas. In this paper, we demonstrate that some recently developed MAPF algorithms apply more broadly than currently believed in the MAPF research community. In particular, we describe the 3D Pipe Routing (PR) problem, which aims at placing collision-free pipes from given start locations to given goal locations in a known 3D environment. The MAPF and PR problems are similar: a solution to a MAPF instance is a set of blocked cells in x-y-t space, while a solution to the corresponding PR instance is a set of blocked cells in x-y-z space. We show how to use this similarity to apply several recently developed MAPF algorithms to the PR problem, and discuss their performance on real-world PR instances. This opens up a new direction of industrial relevance for the MAPF research community.

Introduction
The 3D Pipe Routing (PR) problem is a common industrial problem that appears when designing the layout of industrial plants, such as natural gas processing stations, water treatment facilities, and the power plants used in ships and submarines. Designing the layout of such a plant requires finding 3D coordinates for every piece of equipment in the plant (the equipment allocation problem), and finding a 3D route for every pipe that connects two pieces of equipment (the PR problem). The aim is to minimize the total cost of the plant (which can run into multi-billion dollar budgets), while ensuring safety and correct functionality. Figure 1 shows a layout for part of a natural gas plant.

Differences in the quality of the final layout can have a very significant impact on the cost of these plants, including the cost of the pipes and associated support structures, which are known to take the largest share: up to 80% of the purchased equipment cost or 20% of the fixed-capital investment (Peters and Timmerhaus 2004). However, finding high-quality plant layouts is remarkably difficult due to the size of these plants and the complexity of the associated constraints. As a result, layouts are still designed manually, taking multiple engineers many months (or even years) to complete. This process is inefficient, costly, and the results may vary in quality, since they largely depend on the experience of the piping and layout engineers.

In this work, we focus on the PR problem, which shares many similarities with 2D Multi-Agent Path Finding (MAPF). A solution to the 2D MAPF problem is a set of collision-free paths in x-y-t space where t is the time dimension. A solution to the PR problem is a set of collision-free paths in x-y-z space. We exploit the similarities to apply recently developed priority-based MAPF algorithms to the PR problem, with the goal of finding good solutions quickly. We consider a variety of challenging real-world PR problems provided by our industry partner, which have up to several hundred pipes. Our approach improves convincingly on a currently state-of-the-art PR technique from (Belov et al. 2017), both in terms of success rate (we route more pipes) and solution quality (we achieve smaller costs).
The Real-World Pipe Routing (PR) Problem

Current research into automatic plant layout commonly divides it into two phases. The first phase performs equipment allocation, that is, finds 3D positions for all necessary equipment of the plant with respect to a set of operational constraints. In this phase, the cost of the pipes is approximated using rough measures, such as Manhattan distances. The second phase solves the PR problem directly, by finding 3D routes for the pipes that connect the input and output nozzles of the already allocated equipment.

PR is made challenging by a variety of constraints, each of which is necessary to satisfy operational, maintenance, and safety criteria. In order to satisfy all these different constraints in a reasonable amount of time, most optimisation methods for solving the PR problem route pipes one by one (i.e., sequentially) according to a given order. In this section, we describe the most important features of the PR problem and the various simplifications made by the PlantLayout system (Belov et al. 2018), a prototype industrial software that provides us with a reference model.

The Routing Landscape. Obstacles can have any geometric shape. PlantLayout approximates them by cuboids. We distinguish between physical obstacles, such as equipment, and logical ones, such as maintenance access zones.

Pipe Diameter. Pipes have a certain diameter and require some minimal distance to other objects. PlantLayout accurately models this.

Circular Bends and Non-Axis-Parallel Segments. Pipes can bend with a certain radius. Moreover, although in practice most pipe segments are parallel to one of the coordinate axes, a few are not. PlantLayout assumes pointed bends and axis-parallel segments. However, to account for the bend radius of (usually) \(1.5D\), where \(D\) is the pipe’s diameter, each segment must be at least 3D long (1.5D for nozzle segments).

Support Costs. A pipe has to be supported either by the ground or by some other equipment. The further it is from the supporting object or the ground, the higher the support costs. Moreover, in practice, several pipes can be supported together when their routes are close. PlantLayout approximates the cost aspect by constraining every bend to be at most 3 meters away from at least one equipment item or the ground, and adding a height penalty for each bend when it is not in a supporting structure, such as a pipe rack.

Bend Costs and Penalties. Bends are expensive in terms of construction and operation. PlantLayout includes penalty terms in the objective function of the routing method to account for this.

Stress and Flexibility Analysis. Pipes may contract and expand due to temperature changes in the environment and the materials they transport. This poses stress on the pipe, which needs to be accounted for using a stress and flexibility analysis. There are several methods differing in their complexity and precision (ASME International 2017). While PlantLayout allows for an approximate control of pipe stress during optimization, it was not used in this study.

Existing Methods for the PR Problem

In practice, the PR problem is solved iteratively under considerations from various domains, including further operational and construction aspects not listed above. In particular, there is a strong emphasis on the cooperative allocation of pipes (e.g., reusing support structures), or pipes and equipment together, going far beyond conflict-free routing only. These aspects have not been studied enough, and we leave them for future work. Below, we review methods known in the research literature.

Most research into plant layout design focuses on the equipment allocation phase (e.g., Xu and Papageorgiou; Xu and Papageorgiou 2007; 2009). For small instances of the problem, Sakti et al. (2016) successfully apply a satisfaction (rather than optimisation) method that simultaneously allocates equipment and routes the pipes. Authors consider 10 equipment pieces and up to 15 pipes (4 segments per pipe on average). However, their method fails to find solutions for instances with as few as 8 pipes. Realistic plants are much larger. For example, Belov et al. (2018) consider an instance of a Liquefied Natural Gas (LNG) plant, with 76 equipment pieces and 85 pipes within a 1250x500x200 grid graph. Instances with all equipment in the plant have hundreds of equipment pieces and pipes.

A variety of search-based methods focus specifically on PR. Some of these rely on meta-heuristics, such as the ant-colony evolutionary algorithms considered by Furuholmen et al. (2010) and Jiang et al. (2015). Another popular approach involves prioritised planning (Cao et al. 2018), wherein the pipes are ordered according to some fixed priority and routed in order. Each higher-priority pipe then becomes an obstacle for all lower-priority pipes.

Another approach to PR, similar to that of PlantLayout, involves the use of combinatorial optimisation. Guirardello and Swaney (2005) describe a detailed mixed-integer programming (MIP) model for solving phase one of the plant design layout problem, and they give a general overview of a network-flow MIP model that can solve phase two (i.e., PR). Their PR method involves the construction of a reduced connection graph that limits the possible routes of the pipes. The graph is used to route pipes one by one, as the authors suggest that simultaneous routing is too costly. Since this work gives only a high-level overview, some details are omitted, including the construction of the graph. One approach to building connection graphs is considered by de Berg et al. (1992), who present a higher-dimensional rectilinear shortest path model that considers bend costs. Another approach is given by Zhu and Latombe (1991), using cuboid free space decomposition and is also applied to PR. However, even if these methods are used, it is not clear how Guirardello and Swaney (2005) resolve situations where pipes can interfere with each other. (Guirardello and Swaney 2005 talk about “some tuning by hand” which might be required for these cases.)

In this work, we employ a general PR method first considered by Belov et al. (2017). This approach relies on a high-level model of the PR problem written in the solver-independent language MiniZinc (Nethercote et al. 2007). The model can be solved with a variety of optimisation tech-
ologies, including MIP, and it allows routing several pipes simultaneously, although this approach becomes quickly intractable. We apply this method as a single-agent solver in the context of Priority-Based Search (Ma et al. 2019), a recent technique developed for Multi-Agent Path Finding (MAPF). We show that these two approaches (prioritised search at the high-level and realistic single PR at the low-level) can find state-of-the-art solutions to industrial problems with up to hundreds of pipes.

Prioritised Planning

Definition 1. We call an assignment of routes to pipes a plan. A complete plan is a plan where all pipes have routes. A maximal plan is either complete, or not allowing further routes under the active priority ordering (because of lack of space). A feasible plan is a conflict-free maximal plan.

Prioritised planning (Erdmann and Lozano-Pérez 1987) is a broad family of multi-agent coordination techniques which share the same basic principles. First, agents are planned for sequentially, each from its start location to its target location. Second, during pathfinding, each agent is required to avoid all other agents previously planned for.

Although incomplete in general (see Figure 2), prioritised planning algorithms appear widely in the literature and have been used in a variety of different contexts, including PR (Guirardello and Swayne 2005; Belov et al. 2017; Cao et al. 2018). One of the main advantages is performance: with \( k \) agents to coordinate, a feasible plan can be available after just \( k \) single-agent searches. This has allowed practitioners to develop scalable approaches to otherwise intractable problems with hundreds of moving agents (Silver 2005). The cost of a plan can depend on the priority ordering, and there are \( k! \) possible orders in total. Various ordering heuristics have been considered, including for PR (Belov et al. 2017). However, choosing a good set of priorities remains a challenging problem sometimes left to human experts (Cao et al. 2018).

**Priority-Based Search (PBS)**

PBS (Ma et al. 2019) is a recent coordination algorithm that computes priority orderings online instead of a priori. We give herein a brief description, since the algorithm is central to our work. Note that in the below descriptions each agent is a pipe and each path is a trajectory in three dimensions: from one distinguished nozzle called the start to another nozzle called the target. While the major difference to (Ma et al. 2019) is the low-level subproblem, which is the PR problem, we made some other changes highlighted below.

Search Space. PBS is a branch-and-bound algorithm. It searches a binary Conflict Tree (CT), where each node corresponds to a complete plan. The root of the CT tree corresponds to a plan where every agent ignores all the rest and follows one of its own individually optimal shortest paths. Notice that none of the agents have priorities at this point.

Conflicts. When PBS expands a CT node, it first checks whether the proposed plan contains any collisions among the agents. Such collisions are called conflicts. If there are conflicts, PBS chooses a conflict according to some policy and then works to resolve the impasse by branching the current plan.

Branching. To resolve a conflict between agents \( i \) and \( j \), PBS generates two new child CT nodes: one with added constraint \((i \prec j)\) indicating that agent \( i \) has higher priority than agent \( j \), and the other one with added constraint \((j \prec i)\), which gives \( j \) higher priority. The conflict is resolved at each child node by computing a new individually optimal path for the lower priority agent. The other agents all retain their current paths in both child nodes.

Pathfinding. To plan a new individually optimal path, we apply the optimisation method from (Belov et al. 2017). This method uses a high-level MiniZinc model of the PR problem, which specifies all associated costs and operational constraints. We instantiate the model by adding as obstacles the set of all plant equipment, together with the path of each pipe that corresponds to a higher priority agent. We also introduce a set of additional obstacles, called fake nozzles, which correspond to the shortest possible start and target nozzle segments of the pipes that have yet to be routed. In our setting, fake nozzles occupy a length twice the diameter of the corresponding pipe. We use MIP solver Gurobi (Gurobi Optimization 2020) to solve the model, that is, to plan a path that minimises the overall cost computed using the length and diameter of the pipe, the number of bends, and any associated supports.

Tree Traversal. PBS explores the CT by performing a depth-first traversal. The algorithm continues down a given branch, as long as the order specified by the set of constraints that appear on the path from the root CT node to the current CT node is consistent, i.e., has no directed cycles. When looking for a new path for a lower priority agent, the pathfinder can fail. This indicates there is no path for the set of specified constraints. In contrast to (Ma et al. 2019), we allow a certain number of failed pipes, limited by parameter maxMissing, and only backtrack if that number is exceeded. PBS also backtracks when the current node is a feasible plan, to try to improve it.

Termination. The search terminates when a user-specified time limit is reached, or when the search exhausts...
Algorithm 1: PBS for Pipe Routing

Data: Equipment layout and pipes to be routed with no priority set
Input: conflictPolicy, maxMissing, timeOut
1 Root.plan, Root.constraints ← IndependentRouting(), ∅
2 STACK ← {Root}
3 bestPlan ← ∅
4 while STACK ≠ ∅ and (∼timeOut or bestPlan = ∅) do
5     N ← pop(STACK)  // Depth-first search
6     if Quality(N.plan) ≺ Quality(bestPlan) then
7         bestPlan ← N.plan
8     if N.plan has no conflicts then
9         newNodes ← N.plan
10    else
11        nodes ← Branch(N, conflictPolicy, maxMissing)
12        Insert nodes into STACK in non-increasing order of their quality
13 Function Branch(N, conflictPolicy, maxMissing) :
14    C ← GetConflictPair(N.plan, conflictPolicy)
15    newNodes ← ∅
16    foreach p involved in C (let the other pipe be q) do
17        if N.constraints are consistent with (p ∼ q) then
18            child ← empty node
19            child.plan ← N.plan
20            child.constraints ← N.constraints ∪ {p ∼ q}
21            child.plan.q ← GetPath(child, q)
22            if child.plan.q ≠ ∅ or maxMissing ≥ NumberOfFailed(child.plan) then
23                newNodes ← newNodes ∪ {child}
24        return newNodes
25

Function GetPath(N, p) looks for an individually optimal route for pipe p that avoids all equipment, fake nozzles, and all locations occupied by pipes with higher priority (as defined by the current tentative plan at node N). When no feasible route exists, GetPath(N, p) returns ∅ and the current plan has one more missing pipe.

Function IndependentRouting() returns a set of independently optimal routes for all pipes/agents. Equipment must still be avoided, as well as fake nozzles.

Function Quality(S) returns, for a partial or complete plan S, a tuple \( (num\_missing, c) \). Here, \( num\_missing \) is the number of pipes that could not be routed, and \( c \) is the total cost of the routed pipes. When comparing two plans, we do so lexicographically using the precedence operators \( ∼ \) and \( ≤ \), where \( S_1 ∼ S_2 (S_1 ≤ S_2) \) means the plan \( S_1 \) is better (not worse) than \( S_2 \). We define Quality(∅) \( ≡ (∞, ∞) \).

Function GetConflictPair(S, conflictPolicy) picks a pair of conflicting pipes from \( S \), using the conflict policy conflictPolicy. We consider two conflict policies implemented as distributions:

1. Uniform distribution.
2. Weights proportional to the costs of the two conflicting pipes.

Thus, both conflict policies are random, and any conflict can be selected, but conflict policy 2 prefers those with ‘big’ pipes involved.

Sampling the Conflict Tree (CT)

In addition to PBS, we consider several sampling-based approaches to explore the PBS CT.

FixOrder is our reference algorithm from (Belov et al. 2017). It routes pipes sequentially according to an a priori fixed priority ordering based on non-increasing estimated total costs.

OneDive is a version of PBS that explores a single branch of the CT. Algorithm 2 gives a pseudo-code description. Given an infeasible plan, OneDive fixes the paths of some agents (deemed higher priority according to the policy) and replans all remaining agents (deemed lower priority) whose paths are in conflict with the higher priority set. It exits prematurely whenever it discovers that it cannot improve the reference plan provided (Line 10). OneDive employs the following additional two functions.

Function SelectRouteToFix(C, fixPolicy) uses, for a given conflict C = (i, j), the given fixing policy fixPolicy to return either pipe i or pipe j with some probability. We again consider two distributions:
1. Uniform distribution (both pipes equally preferred).

2. Weights proportional to the costs of the two pipes.

Similar to the conflict policies, any pipe can be selected with both fixing policies, but fixing policy 2 prefers larger/costlier pipes.

Function FindConflictedRoutes(routes, r) returns, for a given route r in routes, all other routes conflicting with r.

Randomized Restart (RR) performs, similar to (Cohen et al. 2018), OneDive at least once, with given conflict and pipe selection policies, and returns a best found plan, as shown in Algorithm 3.

Hill Climbing (HC) accepts an initial feasible plan. Then, a certain portion of the routes are discarded, and the agents are replanned using OneDive, as described in Algorithm 4. We distinguish two versions: HC(OneDive) and HC(FixOrder), where the initial solution is obtained by OneDive or FixOrder, respectively. HC depends on the following function:

Function SelectUnfixRoutes(routes, destructionPercent) selects routes to be rerouted in the current Hill Climbing iteration. First, any agents without currently assigned paths (i.e., missing pipes) are selected, and then more routes are selected at random until the destructionPercent percentage of routes are selected. The selected pipes are rerouted using OneDive (with the non-selected routes being fixed as obstacles). This procedure is performed iteratively. It shares similarities with Large Neighbourhood Search (Shaw 1998), a meta-heuristic approach popularly used for vehicle routing problems.

### Experiments

We test the performance of the algorithms on both industry-size instances with up to 207 pipes, as well as on smaller specially constructed synthetic instances with up to a few dozen pipes. We benchmark the efficiency of the heuristic randomization policies, as well as the overall runtime profiles of the algorithms.

### Instances

Our synthetic instances were constructed to challenge the algorithms due to their small congested spaces with a high density of pipes. The number of pipes ranges from 8 to 36 per instance, while the pipe diameter is 1000mm. We have a Small instance with just 8 pipes, and four Medium instances, three of them with 23 pipes and the last one with 36 pipes.

We also have two industrial instances from plants intended to extract Liquefied Natural Gas. Instance AGRU (Acid Gas Removal Unit) has 66 pipes, with diameters ranging from 50mm to 750mm. Instance LNG Train includes 207 pipes with diameters ranging from 50mm to 1500mm.

Details for all instances, including the associated runtime limits used for the experiments, are specified in Table 1. The 3D layouts of the instances are shown in Figure 3. The Medium 1, 2 and 3 instances have the same set of equipment and connections but different equipment layout.

### Algorithms and Implementation

We test several high-level PR algorithms with various options. Our reference in all cases is FixOrder, the fixed priority ordering algorithm used in (Belov et al. 2017), where pipes are routed sequentially according to their non-increasing estimated total cost.

Algorithm RR(c, f) represents PBS with randomised restarts, where parameters c and f indicate specific conflict and fixing policies. Algorithm HC represents Hill Climbing with conflictPolicy = 2 and destructionPercent = 50%. We distinguish HC(OneDive) and HC(FixOrder), depending on the initial plan provided. We also run the PBS algorithm with maxMissing = 0, which we denote as PBS, and with maxMissing = ∞, which we denote as PBS-MP. Recall that the maxMissing variable indicates how many pipes can be left without any route in a tentative plan.

<table>
<thead>
<tr>
<th>Instance</th>
<th>EP</th>
<th>LO</th>
<th>P</th>
<th>BB</th>
<th>ρ</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>11 × 10 × 12</td>
<td>15.5%</td>
<td>960</td>
</tr>
<tr>
<td>Medium 1</td>
<td>12</td>
<td>12</td>
<td>23</td>
<td>28 × 18.4 × 9.0</td>
<td>20.8%</td>
<td>2760</td>
</tr>
<tr>
<td>Medium 2</td>
<td>12</td>
<td>12</td>
<td>23</td>
<td>19 × 13.3 × 18</td>
<td>21.5%</td>
<td>2760</td>
</tr>
<tr>
<td>Medium 3</td>
<td>12</td>
<td>12</td>
<td>23</td>
<td>18.4 × 11 × 28</td>
<td>17.3%</td>
<td>2760</td>
</tr>
<tr>
<td>Medium 4</td>
<td>12</td>
<td>12</td>
<td>36</td>
<td>19 × 11 × 24</td>
<td>19.1%</td>
<td>4320</td>
</tr>
<tr>
<td>AGRU</td>
<td>53</td>
<td>24</td>
<td>66</td>
<td>86 × 49 × 40</td>
<td>13.9%</td>
<td>7920</td>
</tr>
<tr>
<td>LNG Train</td>
<td>166</td>
<td>47</td>
<td>207</td>
<td>192 × 60 × 60</td>
<td>21.8%</td>
<td>24840</td>
</tr>
</tbody>
</table>

Table 1: Instance details. EP is the number of equipment pieces, LO is the number of logical obstacles, P is number of pipes to be routed, BB is bounding box of the routing landscape, in meters, ρ is the density of space occupation by obstacles, in percent, and t is the runtime limit, in seconds, allowed per instance for any algorithm, chosen as 120s times the number of pipes.
Figure 3: Layouts for the different instances. Subfigures (a)–(d) show the layouts for the Small, Medium 4, AGRU and LNG Train instances, respectively, while Subfigures (e)–(g) show those for the Medium 1-3 instances. Blue boxes in the Small instance are boundary access zones, while yellow boxes are placeholders for “skirts” (support basements).

For low-level routing, we use the algorithm of Belov et al. (2017), executed with MiniZinc 2.4.2 and the MIP backend Gurobi 9.0.1. For each pipe, only 6 bends are initially allowed during the search, with this number being increased if no route is found. The most difficult part of this algorithm resides in the non-overlapping constraints among pipe segments and obstacles. Each pipe is routed with a runtime limit of 180 seconds after the first feasible route (the runtime until then is not limited). This algorithm does not represent the routing landscape as a grid, as is common for MAPF. Instead, it uses variables to specify bend locations. These variables use a space discretisation with a granularity of 1cm in our experiments.

All PBS algorithms are implemented in C++. Our machine has a 3.4GHz Intel i7-4770 processor with 16GB 1600MHz RAM and Ubuntu Linux 18.04. For repeatability, we used the same random seed across all algorithms and instances.

**Evaluation Criteria**

For the performance evaluation, we focus on two aspects:

- **Plan quality** measures the effectiveness by the number of missing pipes (as compared to independent routing) and the total cost. The cost is represented by a **cost gap** to the cost of independent routing. For example, suppose that the total cost of independent routing is X and the total cost of a final layout found by an algorithm is Y. Then, we evaluate the cost gap as $Y/X - 1$. Note that this cost gap is only informative when the number of successfully routed pipes is the same as in the independent routing.

<table>
<thead>
<tr>
<th>Instance</th>
<th>P</th>
<th>IP</th>
<th>RR(1,2)</th>
<th>RR(2,2)</th>
<th>RR(2,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>13.1</td>
<td>0</td>
</tr>
<tr>
<td>Medium 4</td>
<td>36</td>
<td>36</td>
<td>2</td>
<td>9.4</td>
<td>11.6</td>
</tr>
</tbody>
</table>

- **Runtime** measures the efficiency. We limit the total runtime for each algorithm. For algorithm performance reporting, we use the time points when improved plans are found. We focus on the best plan found at any point in time. The runtime limits per instance for all algorithms are provided in Table 1.

**Heuristic Methods and Randomization Policies**

First, we benchmark the conflict and fixing policies using algorithm $RR(c, f)$. To do this, we compare each policy against a uniform random alternative.

**Conflict policy (c)**. We use policy 1 to denote the uniform random approach, and policy 2 to denote the randomized...
Comparison of Algorithms

Figure 4 shows performance results for the algorithms on all instances, except for the Medium 4 instance. The legend is given in Figure 4(c). The X axis of each plot shows the runtime in seconds. The Y axis in the upper part of each plot shows the cost gap, while, in the lower part, it shows the number of missing pipes \( M \), as compared to independent routing. The cost gaps of two plans can only be meaningfully compared if they have an equal number of missing pipes.

**Synthetic Instances.** All algorithms find plans for the Small instance with a complete set of pipes (their lower graphs converge to height 0). The best gap of 1.3% is achieved by \( \text{RR}(2,2) \). For all instances, most algorithms improve their initial number of missing pipes \( M \). While the plans get closer to being complete when \( M \) improves, the cost gap can increase. For example, for the Medium 2 instance, most heuristics start with \( M \geq 2 \), including \( \text{FixOrder} \), which produces just a single plan. All other algorithms, except for PBS-MP, improve that number to 1 or 0 pipes missing.

**Industry Instances.** For the AGRU instance, \( \text{FixOrder} \) failed to route two pipes but all other algorithms succeeded to route all pipes. For the LNG Train instance, all algorithms succeeded to route all pipes. For both instances, the best final cost was smaller than \( \text{FixOrder} \)'s cost despite having more pipes for the AGRU.

Overall, we observe that, in many cases, the new algorithms improve upon \( \text{FixOrder} \) with respect to the number of successfully routed pipes or the cost. Interestingly, even the first plans obtained by the algorithms are mostly better. This comes at a price since they are more runtime intensive than \( \text{FixOrder} \). Subsequent iterations or branching usually
further improves the results. Table 3 provides more data on the best final plans, including for the Medium 4 instance.

**Significance.** It should be noted that improvements of several percent in cost can mean savings of millions of dollars in practice, due to the high costs of plant construction and operation. The new algorithms achieve these improvements with longer runtimes compared to a reference method. Unlike MAPF, where deliberation time is limited, plant layout is a multi-year problem and runtimes measured on the scale of minutes and even hours are not prohibitive.

**Conclusion**

We consider 3D Pipe Routing (PR), an important and challenging industrial problem which appears in the context of designing industrial plant equipment. To solve PR we develop a range of heuristic techniques in the family of Priority-Based Search (PBS): a recent prioritised planning method developed for Multi-Agent Pathfinding (MAPF) (Ma et al. 2019).

Given enough time, we find that PBS can often produce best known solutions to challenging industrial instances with up to hundreds of pipes. To compute solutions faster we also consider a number of local-search variants including randomised restarts, for sampling the PBS conflict tree, and hill climbing, where PBS starts from and attempts to improve a best known candidate plan. Experimental results show that these approaches can improve not only performance vs PBS but also plan quality, routing more pipes faster and at lower cost.

A variety of directions exist for future work e.g., the inclusion of meta-heuristics which can allow the search to escape local optima and/or guide the search toward more promising plans. In practice, pipes are also often routed together, reusing support structures. Thus, it can be fruitful to combine equipment allocation and pipe routing so as to avoid the issue of unroutable pipes. Although current methods are faster than manual routing there exists substantial scope for improvement.

**Acknowledgements**

The research at Monash University was funded by Woodside Energy Ltd. We thank all our Woodside collaborators, particularly Solomon Faka, for the many useful discussions, as well as for the enlightening visit to their LNG plant.

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, 1817189, 1837779, and 1935721.

**References**


