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 Wei[1]
[1]Monash University, [2]Australian National University, [3]University 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.

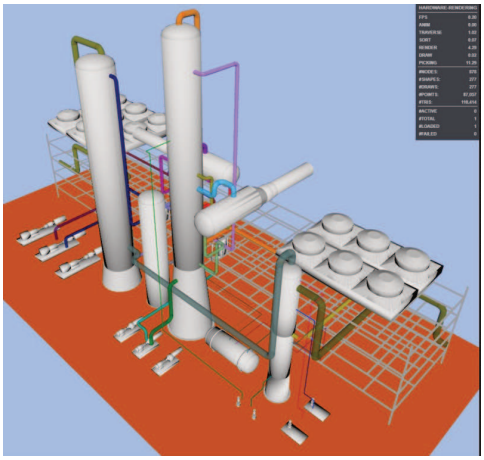

Figure 1: Example layout for the acid gas removal unit in a liquefied natural gas plant.

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 similarites 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 part 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-

nologies, 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 all share the same basic principles. First, agents are planned for sequentially, each from its start location to its target location. Second, during pathfinding search, 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 Swaney 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.

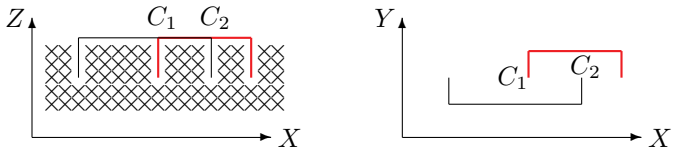

Figure 2: Prioritised planning example. For the same layout, left is the XZ-projection (side-view) and right is the XY-projection (top-view). Blocked cells are marked with $\times$ on the left. Assume this obstacle structure is replicated along the Y direction. There are two pipes, red and black, with their start-goal locations in holes 1-3 and 2-4 (counting from the left), respectively. C_1 and C_2 mark conflict points. Prioritised planning always fails in this case, irrespective of the ordering. An optimal plan (made clear on the right) has the two pipes swap priorities at the conflict points.

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 every 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 **if** *N.plan* has no conflicts **then**
8 *bestPlan* ← *N.plan*
9 **else**
10 *nodes* ← Branch(*N, conflictPolicy, maxMissing*)
11 Insert *nodes* into *STACK* in non-increasing order of their quality
12 **Function** `Branch` (*N, conflictPolicy, maxMissing*) **:**
13 $\mathcal{C}$ ← GetConflictPair(*N.plan*, *conflictPolicy*)
14 *newNodes* ← ∅
15 **foreach** *p* involved in $\mathcal{C}$ (let the other pipe be *q*) **do**
16 **if** *N.constraints* are consistent with $(p \prec q)$ **then**
17 *child* ← empty node
18 *child.plan* ← *N.plan*
19 *child.constraints* ← *N.constraints* ∪ $\{p \prec q\}$
20 *child.plan.q* ← GetPath(*child*, *q*)
21 **if** *child.plan.q* ≠ ∅ **or** *maxMissing* ≥ NumberFailed(*child.plan*) **then**
22 *newNodes* ← *newNodes* ∪ {*child*}
23 **return** *newNodes*

Algorithm 2: Function OneDive

Input: *plan, routesFixed, conflictPolicy, fixPolicy, bestPlan*

1 **while** *plan* has any conflicts **do**
2 $\mathcal{C}$ ← GetConflictPair(*plan*, *conflictPolicy*)
3 *routeToFix* ← SelectRouteToFix($\mathcal{C}$, *fixPolicy*)
4 *routesFixed* ← *routesFixed* ∪ *routeToFix*
5 *routesToReroute* ← FindConflictedRoutes(*plan*, *routeToFix*)
6 *plan* ← *plan* \ *routesToReroute*
7 **for** *route* ∈ *routesToReroute* **do**
8 *plan* ← *plan* ∪ GetPath(*routesFixed, route*)
9 **if** Quality(*bestPlan*) ⪯ Quality(*plan*) **then**
10 **return** ∅ // Cannot improve *bestPlan*
11 **return** $plan$

the tree.

Differences to the PBS of (Ma et al. 2019). Algorithm 1 provides a detailed pseudo-code description of our proposed PBS implementation. While very similar in principle to (Ma et al. 2019), it has a number of important differences. First, we consider feasible plans with unrouted/missing pipes. This is because we do not know if all pipes can be successfully routed in a conflict-free way. Parameter *maxMissing* is the largest allowed number of missing pipes. Correspondingly, the *quality* of a plan is measured not only by its total cost, but primarily by the number of successfully routed pipes. Second, we do not terminate after the first conflict-free plan, but try to improve it. In doing this, we store the best feasible plan computed thus far as an incumbent (Line 8). We use the incumbent to bound the quality of the current node and never explore any branch where the quality of the candidate plan is equal to or worse than the incumbent (Line 6); that is, where the number of missing pipes is not smaller and, if equal, the cost is not smaller. And third, we parameterise the algorithm with a conflict policy, that is, a heuristic that helps PBS to decide which conflict it should branch on next.

We now describe the role of several important PBS functions, which we again adapt for PR.

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 $\mathcal{S}$, 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 $\prec$ and $\preceq$, where $S_1 \prec S_2$ ($S_1 \preceq S_2$) means the plan S_1 is better (not worse) than S_2. We define $Quality(\emptyset) \equiv (\infty, \infty)$.

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*($\mathcal{C}$, *fixPolicy*) uses, for a given conflict $\mathcal{C} = (i, j)$, the given fixing policy *fixPolicy* to return either pipe i or pipe j with some probability. We again consider two distributions:

Algorithm 3: Randomized Restart

Input: *timeOut*, *conflictPolicy*, *fixPolicy*

routes ← IndependentRouting()
bestRoutes ← ∅
repeat
  currentRoutes ← OneDive(*routes*, ∅, *conflictPolicy*, *fixPolicy*, *bestRoutes*)
  if Quality(*currentRoutes*) ≺ Quality(*bestRoutes*) **then**
    bestRoutes ← *currentRoutes*
until *timeOut*;
return *bestRoutes*

Algorithm 4: Hill Climbing

Input: *bestRoutes*, *timeOut*, *destructionPercent*, *conflictPolicy*, *fixPolicy*

routes ← IndependentRouting()
while !*timeOut* **do**
  routesToRepair ← SelectUnfixRoutes(*bestRoutes*, *destructionPercent*)
  currentRoutes ← OneDive(*routes*, *bestRoutes* \ *routesToRepair*, *conflictPolicy*, *fixPolicy*, *bestRoutes*)
  if Quality(*currentRoutes*) ≺ Quality(*bestRoutes*) **then**
    bestRoutes ← *currentRoutes*
return *bestRoutes*

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.

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 $\bar{t}$ is the runtime limit, in seconds, allowed per instance for any algorithm, chosen as 120s times the number of pipes.

Instance	*EP*	*LO*	*P*	*BB*	ρ	$\bar{t}$
Small	4	7	8	$11 \times 10 \times 12$	15.5%	960
Medium 1	12	12	23	$28 \times 18.4 \times 9.0$	20.8%	2760
Medium 2	12	12	23	$19 \times 13.3 \times 18$	21.5%	2760
Medium 3	12	12	23	$18.4 \times 11 \times 28$	17.3%	2760
Medium 4	12	12	36	$19 \times 11 \times 24$	19.1%	4320
AGRU	53	24	66	$86 \times 49 \times 40$	13.9%	7920
LNG Train	166	47	207	$192 \times 60 \times 60$	21.8%	24840

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* $= \infty$, 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.

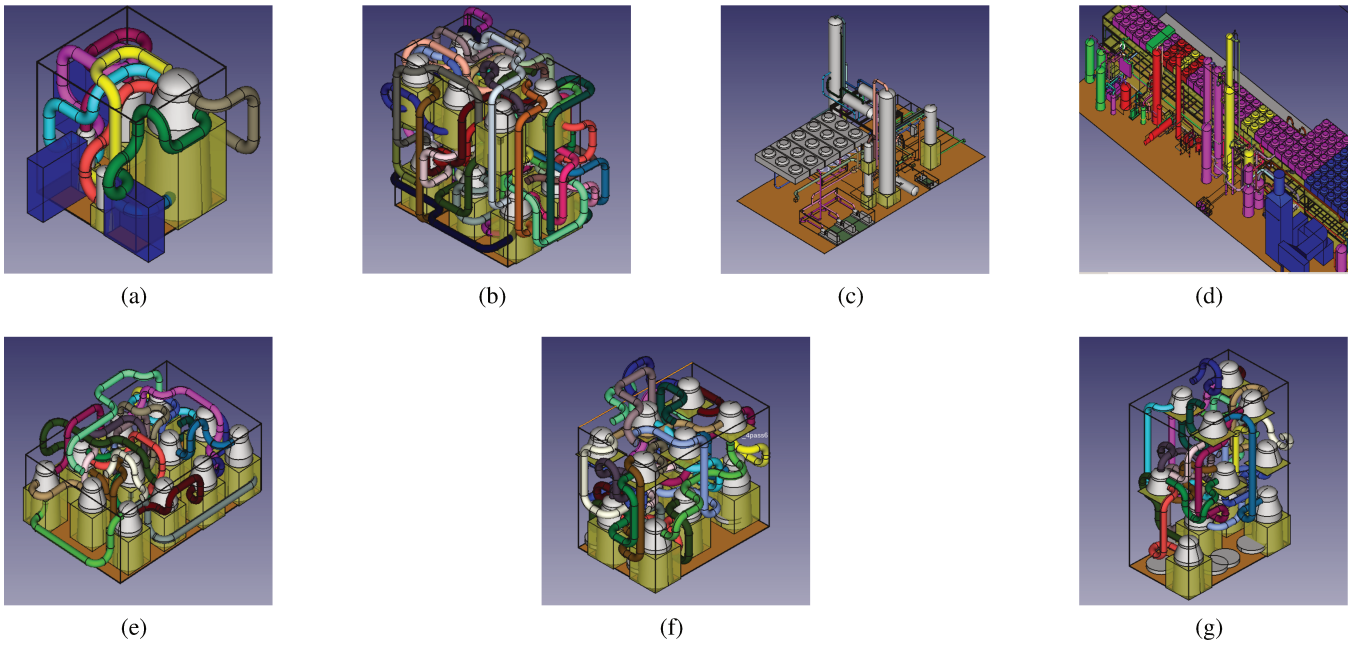

Figure 3: Layouts for the different instances. Subfigures (a)–(d) show the layouts for the Small, Medium 4, AGRU and LNG Train instances, respectively, while Subfigures (e)–(g) show those for the Medium 1-3 instances. Blue boxes in the Small instance are boundary access zones, while yellow boxes are 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 2: Evaluation of the conflict and fixing policies. *P* is the number of pipes in the instance. *IP* is the number of pipes successfully routed with independent routing. *M* and cost gap (%) characterize the best found plan for each algorithm. *M* is the number of missing pipes in a final plan, compared to independent routing. A value of *M*=0 means the plan has the same number of unrouted pipes as independent routing.

			RR(1,2)		**RR**(2,2)		**RR**(2,1)	
Instance	*P*	*IP*	*M*	gap	*M*	gap	*M*	gap
Small	8	8	0	18.9	**0**	**13.1**	0	21.7
Medium 1	23	23	**0**	**19.1**	1	3.9	1	16.1
Medium 4	36	36	2	9.4	1	11.6	**1**	**7.8**
Average			0.67	15.8	**0.67**	**9.5**	0.67	15.2

- **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

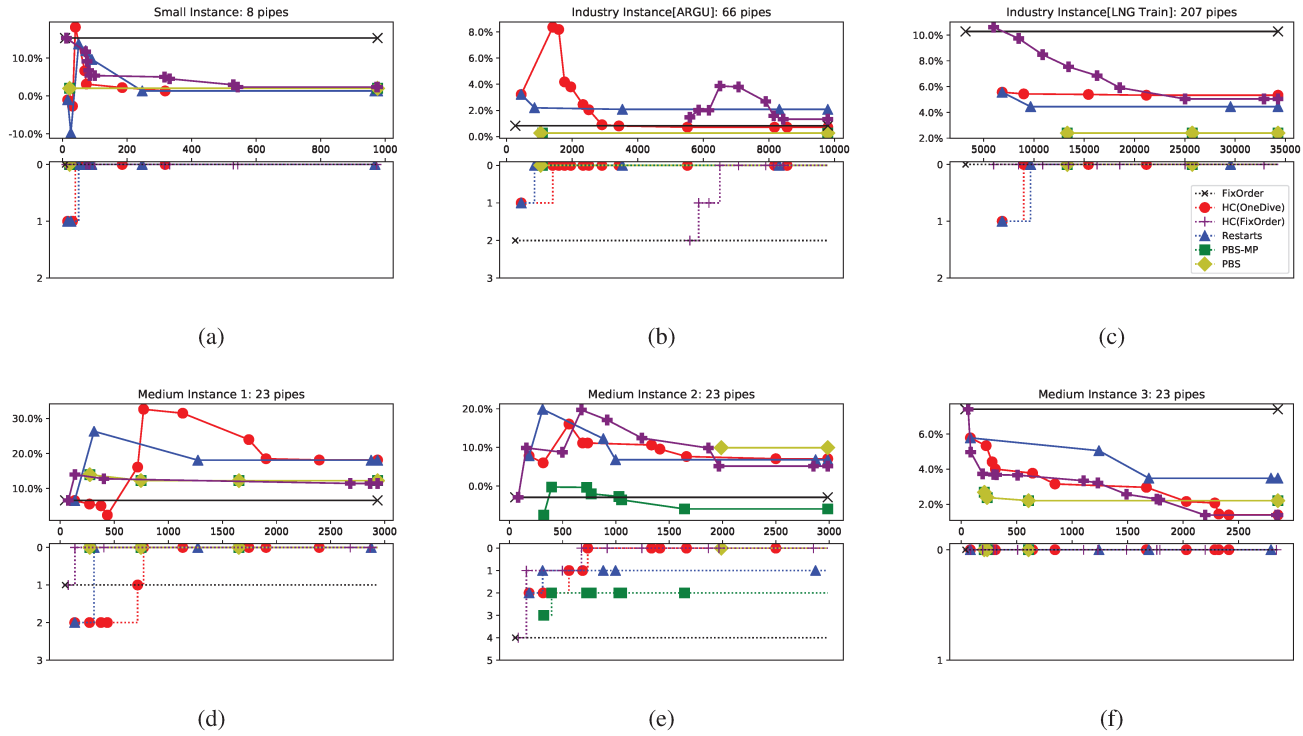

Figure 4: Dynamic performance graphs for all instances, except for the Medium 4 instance. For every instance, the upper part of a plot shows the development of the cost gap of an algorithm's best found plan, while the lower part shows the development of the number of missing pipes M, as compared to independent routing.

selection weighted by the cost of the two conflicting pipes. The fixing policy in this experiment is set to 2. Thus, we compare **RR**(1,2) and **RR**(2,2).

The results appear in Table 2. We used three exemplary instances. For the Medium 1 instance, policy 1 routed one more pipe than policy 2, which is not what we expected. For the Medium 4 instance, while policy 2 missed only one pipe, policy 1 missed two pipes. From the average data, we can conclude that policy 2 obtains a better quality.

Fixing policy (f). We use policy 1 to denote the uniform random approach to choosing which of two conflicting pipes to fix, and policy 2 to denote the randomised selection using the costs of the two pipes as weights. In this experiment, the conflict policy is set to 2, comparing **RR**(2,1) with **RR**(2,2). Results in Table 2 show that there is no difference in terms of the number of routed pipes. However, policy 2 produces plans with smaller cost gaps. This supports our intuition that the cost-based selection is on average better than the uniformly random selection. Based on these results, we use the cost-based policies as default for all algorithms. Thus, from now on, **RR** means **RR(2,2)**.

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 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 **RR** and **HC(OneDive)**. 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 **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, **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 **FixOrder**'s cost despite having more pipes for the AGRU.

Overall, we observe that, in many cases, the new algorithms improve upon **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 **FixOrder**. Subsequent iterations or branching usually

Table 3: Result summary: best final plans. *P* is the number of pipes in the instance. *IP* is the number of pipes successfully routed independently. *M* and cost gap (%) characterize the best found plan for each algorithm. For independent routing and **FixOrder**, t is the runtime in seconds. For the iterative heuristics and PBS, *Dm* is the maximal dive depth, *Ni* the number of iterations and *Nn* the number of nodes.

		Indep		FixOrder			RR			HC(OneDive)			HC(FixOrder)			PBS				PBS-MP			
Instance	*P*	*IP*	t	*M*	gap	t	*M*	gap	*Ni*	*M*	gap	*Ni*	*M*	gap	*Ni*	*M*	gap	*Nn*	*Dm*	*M*	gap	*Nn*	*Dm*
Small	8	8	6	0	15.3	8	**0**	**1.3**	107	**0**	**1.3**	182	0	2.3	167	0	2.0	9	8	0	2.0	9	8
Medium 1	23	23	35	1	6.5	45	0	18	40	0	18.1	47	**0**	**11.4**	58	0	12.2	275	49	0	12.2	275	49
Medium 2	23	23	32	4	-3	55	1	6.8	33	0	7.1	48	**0**	**5.1**	49	0	9.9	296	67	2	-6	316	69
Medium 3	23	23	29	0	7.4	37	0	3.5	42	**0**	**1.4**	77	**0**	**1.4**	80	0	2.2	126	38	0	2.2	126	38
Medium 4	36	36	67	1	19.9	115	2	43.5	10	0	43.9	17	0	36.6	16	**0**	**23.7**	360	114	2	11	332	127
AGRU	66	66	288	2	0.9	268	0	2.1	45	0	0.7	43	0	1.3	36	**0**	**0.3**	58	57	**0**	**0.3**	58	57
LNG Train	207	206	3658	0	10.3	3205	0	4.4	7	0	5.3	9	0	5	9	**0**	**2.4**	607	172	**0**	**2.4**	607	172

further improve 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

ASME International. 2017. ASME B31.3-2016. Standard, process piping.

Belov, G.; Czauderna, T.; Dzaferovic, A.; Garcia de la Banda, M.; Wybrow, M.; and Wallace, M. 2017. An optimization model for 3D pipe routing with flexibility constraints. In *International Conference on Principles and Practice of Constraint Programming*, 321–337.

Belov, G.; Czauderna, T.; Garcia de la Banda, M.; Klapperstueck, M.; Senthooran, I.; Smith, M.; Wybrow, M.; and Wallace, M. 2018. Process plant layout optimization: Equipment allocation. In *International Conference on Principles and Practice of Constraint Programming*, 473–489.

Cao, P.; Fan, Z.; Gao, R. X.; and Tang, J. 2018. Design for additive manufacturing: Optimization of piping network in compact system with enhanced path-finding approach. *Journal of Manufacturing Science and Engineering* 140(8):081013.

Cohen, L.; Koenig, S.; Kumar, T. K. S.; Wagner, G.; Choset, H.; Chan, D.; and Sturtevant, N. 2018. Rapid randomized restarts for multi-agent path finding: Preliminary results. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 1909–1911.

de Berg, M.; van Kreveld, M.; Nilsson, B. J.; and Overmars, M. 1992. Shortest path queries in rectilinear worlds. *International Journal of Computational Geometry & Applications* 2(3):287–309.

Erdmann, M. A., and Lozano-Pérez, T. 1987. On multiple moving objects. *Algorithmica* 2:477–521.

Furuholmen, M.; Glette, K.; Hovin, M.; and Torresen, J. 2010. Evolutionary approaches to the three-dimensional multi-pipe routing problem: A comparative study using direct encodings. In Cowling, P., and Merz, P., eds., *Evolutionary Computation in Combinatorial Optimization*, 71–82.

Guirardello, R., and Swaney, R. E. 2005. Optimization of process plant layout with pipe routing. *Computers & Chemical Engineering* 30(1):99–114.

Gurobi Optimization, L. 2020. Gurobi optimizer reference manual.

Jiang, W.-Y.; Lin, Y.; Chen, M.; and Yu, Y.-Y. 2015. A co-evolutionary improved multi-ant colony optimization for ship multiple and branch pipe route design. *Ocean Engineering* 102:63–70.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with consistent prioritization for multi-agent path finding. In *AAAI Conference on Artificial Intelligence*, 7643–7650.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. Minizinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, 529–543.

Peters, M. S., and Timmerhaus, K. D. 2004. *Plant design and economics for chemical engineers*. New York: McGraw-Hill Book Company, 5th edition.

Sakti, A.; Zeidner, L.; Hadzic, T.; Rock, B. S.; and Quartarone, G. 2016. Constraint programming approach for spatial packaging problem. In *International Conference on the Integration of AI and OR Techniques in Constraint Programming*, 319–328.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, 417–431.

Silver, D. 2005. Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 117–122.

Xu, G., and Papageorgiou, L. G. 2007. A construction-based approach to process plant layout using mixed-integer optimization. *Industrial & Engineering Chemistry Research* 46(1):351–358.

Xu, G., and Papageorgiou, L. G. 2009. Process plant layout using an improvement-type algorithm. *Chemical Engineering Research and Design* 87(6):780–788.

Zhu, D., and Latombe, J. C. 1991. Pipe routing-path planning (with many constraints). In *IEEE International Conference on Robotics and Automation*, 1940–1947.