This is a preprint version of the article: Kulich, M., Woller, D., Carmesin, S., Mansouri, M., and Přeučil, L. (2023). **Where to Place a Pile?**. *Proceedings of 11th European Conference on Mobile Robots*, which has been published in a final form at https://doi.org/10.1109/ECMR59166.2023.10256330.

Where to Place a Pile?

Miroslav Kulich[1], David Woller[1,2], Sarah Carmesin[3], Masoumeh Mansouri[3], and Libor Přeučil[1]

***Abstract*— When planning missions for autonomous machines in real-world scenarios, such as open-pit mining, painting, or harvesting, it is important to consider how the machines will alter the working environment during their operations. Traditional planning methods treat such changes, like piles built during drilling, as constraints given to the planner that depend on the machine's trajectory. The goal is to find a trajectory that satisfies these constraints. However, our approach formulates the planning problem as finding optimal positions for changes, such as piles, along the machine's trajectory. We propose a heuristic solver and provide extensive experimental evaluations.**

I. INTRODUCTION

With increasing levels of autonomy, robots are deployed in more and more complex scenarios. In a mining application, one or more drill machines operate in an open-pit mine to drill blast holes in predetermined targets. After the blast holes are drilled, they are filled with explosive material and detonated, and the ore is processed for mineral extraction. The drill machines can autonomously navigate to the targets, level themselves, drill, and retract. However, the drilling process creates piles of excess material around the hole, which must be cleared before the machine can navigate to the next target. The Drill Pattern Planning Problem (DP^3) [1] for a single drill machine or a fleet of these involves computing a time-optimal plan that ensures the machine(s) can reach each drill target, perform the defined operations, and move away from the target without colliding with obstacles, other machines, or the excess material created during the drilling process.

Mansouri et al. [1] propose a method for solving multi-vehicle DP^3 considering the dimensions of the machines, including the size of the dust guard and jacks used for leveling and the time required to perform each task. The authors break down the problem into sub-problems, identifies interdependency among the sub-problems, and interleaves reasoning within each sub-problem. The approach is further improved and defined as MVRP-DDO (Multi Vehicle Routing with Nonholonomic Constraints and Dense Dynamic Obstacles) in [2]. Carmesin et al. [3] introduce new variants of the Hamiltonian Cycle and Travelling Salesperson problems inspired by the open-pit mining application. Specifically, the authors assume dynamic graphs where edges are deleted or made untraversable depending on the already visited vertices. Besides formal definitions of the problems for such graphs, problems' properties are theoretically analyzed, and two solvers are proposed.

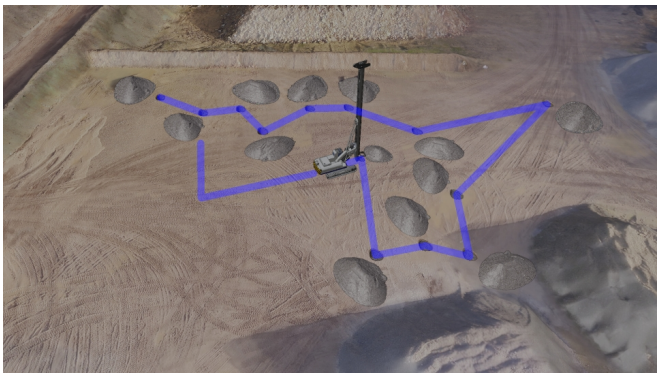


Fig. 1: Drilling scenario.

Another application where heavy vehicles are not allowed to pass already visited areas is autonomous harvesting. In the harvesting application, harvested areas limit the mobility of harvesting machines, hence affecting the reachability among the nodes representing areas to be harvested. Ullrich et al. [4] propose a graph-search planner that searches a directed graph representing the harvesting area. The designed cost function considers the number of passes through individual edges to tackle multiple passing of edges.

The aforementioned approaches assume the constraints caused by drilling/harvesting are predefined, and the planner can only affect the order in which they appear. In [1], [2], for example, piles are placed at the same positions as blast holes. Similarly, the set of edges to be deleted after visiting a vertex is predefined in [3], while edges are removed as they are traversed in [4]. Our approach is different. We assume that piles are built in the vicinity of blast holes, but their exact positions can vary, and the planning algorithm has to decide where to place them. Specifically, a trajectory for a drilling machine is given, and we ask the following questions:

- Can we place piles of a given radius so that the machine's trajectory is not obstructed by them?
- What is the largest radius for which the machine's trajectory is not obstructed by the piles?
- How should the piles be placed to ensure that the machine's path is not obstructed?

[1] Miroslav Kulich, David Woller, and Libor Přeučil are with Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague, Czech Republic {miroslav.kulich, david.woller, libor.preucil}@cvut.cz

[2] David Woller is also with Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo náměstí 13, 121 35 00 Prague, Czech Republic

[3] Sarah Carmesin and Masoumeh Mansouri are with School of Computer Science, University of Birmingham, Edgbaston, B15 2TT Birmingham, United Kingdom sxc1431@student.bham.ac.uk, m.mansouri@bham.ac.uk

979-8-3503-0704-7/23/$31.00 ©2023 IEEE

We formulate two problems related to the above questions formally in Section II and propose a solver for both problems (Section III). The performance of the solver is extensively evaluated through experiments with a specially designed dataset, and the results are discussed in Section IV. The concluding remarks are presented in Section V.

II. PROBLEM FORMULATION

Let $P = \langle p_1, p_2, \dots p_n \rangle$ be a sequence of points in $\mathbb{R}^2$ forming a polygonal path, i.e., a curve consisting of line segments connecting the consecutive points. The *Path Conforming Circles Placement Problem* (PCCP) is to find a set of circles $\mathcal{K} = \{\kappa_i\}_{i \in I}$, where $I = \{1, \dots n\}$, and κ_i is a circle with center c_i and radius r_i, such that:

(C1) the radii of all circles are equal: $r_i = r \quad \forall i \in I$,
(C2) i^{th} point lies on i^{th} circle: $|p_i c_i| = r \quad \forall i \in I$,
(C3) intersection of any two circles is empty: $\kappa_i \cap \kappa_j = \emptyset \quad \forall i, j \in I, i \neq j$,
(C4) intersection of any circle with the path is empty: $\kappa_i \cap P = \emptyset \quad \forall i \in I$, and
(C5) r is maximal.

We say that κ_i is *associated* with p_i.

Let $head_k(P) = \langle p_1, p_2, \dots, p_{k-1} \rangle$ be a head of polygonal path $P = \langle p_1, p_2, \dots p_n \rangle$ and $tail_k(P) = \langle p_k, p_{k+1}, \dots, p_n \rangle$ its tail. The *Weak PCCP* (WPCCP) relaxes condition C4 by allowing for the nonempty intersection of the i^{th} circle with $head_i(P)$. The modified condition for the WCCP is thus:

(4*) $\quad \kappa_i \cap tail_i(P) = \emptyset \quad \forall i \in \langle 1, n \rangle$

Examples of the problem instances and their solutions are shown in Fig. 2. As the WPCCP is less constrained, the maximum radius found for it is higher than the one for the PCCP.

Although the PCCP and WPCCP are new, we can take inspiration from a class of problems seeking the largest empty circle. The classic example of this class is the Largest Empty Circle Problem (LEC) which consists in finding the largest circle C centered in the convex hull of a set of points such that no point lies in the interior of the circle. Shamos [5] propose an algorithm solving the LEC based on an effective search of Voronoi diagrams. Thoussaint [6] subsequently corrected the algorithm showing that Shamos made a wrong assumption about the intersection of a convex hull with a Voronoi diagram, while [7] further improved the algorithm complexity to $O(n[h \log n])$, where n is the number of points and h is the number of convex hull edges. The query variant of the LEC is addressed in [8]. The aim is to preprocess the input points to identify the largest empty circle efficiently. Finally, Augustine et al. [9] address the constrained variant where the circle has to be centered on a given line. All the aforementioned formulations search for a *single* circle while we seek a set of circles. This makes our formulation novel and challenging.

III. APPROACH

In this section, we introduce a solver for both PCCP and WPCCP. We start with the description of a general structure which is the same for both problems. The next subsections III-A to III-C then detail the individual parts of the algorithm. Finally, subsection III-D introduces modifications for the WPCCP.

The algorithm shown in Alg. 1 is motivated by the bisection method [10]. It starts with the estimation of the lower and upper bound of the radius (lines 1 and 2). Then, the bounds are modified by the iterative procedure (lines 3-10). At each iteration, the interval between the bounds is split into two halves by computing the midpoint radius (line 4) and finding the optimal placement of circles with this radius fixed (line 5). If the found placement is valid, the lower bound is replaced by the radius (line 7), and the solution is stored. If the placement is invalid, the upper bound is replaced by the radius (line 8). The process stops when the upper and lower bound difference is below the predefined limit. The stored solution and radius are returned then (line 11).

Algorithm 1: Interval bisection algorithm for the PCCP/WPCCP.

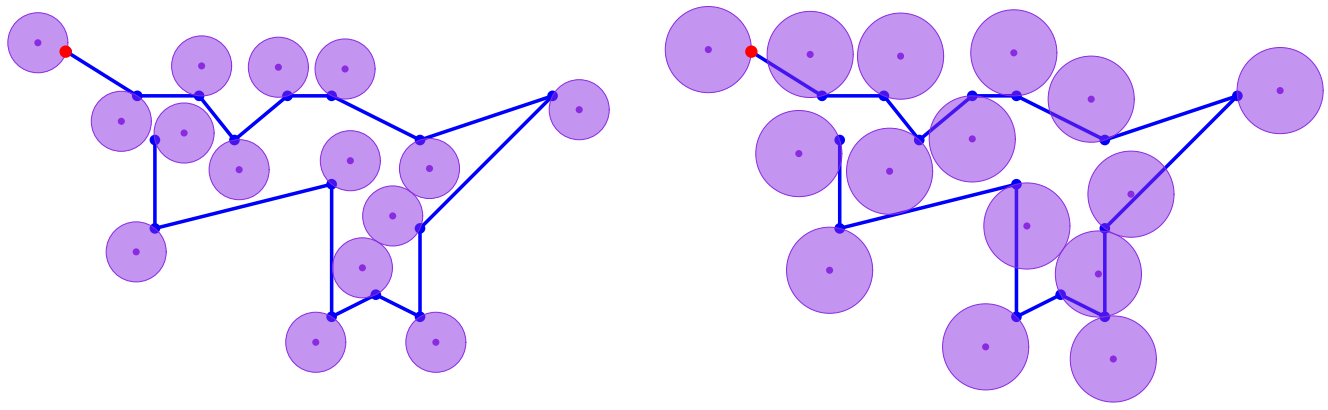
Input: $\mathcal{C}$ – set of cells

```
1  lb ← lower_bound()
2  ub ← upper_bound()
3  while (ub − lb) < ϵ do
4      radius ← (lb+ub)/2
5      (P, valid) ← find_placement(radius)
6      if valid then
7          lb ← radius
8          (P_best, radius_best) ← (P, radius)
9      else
10         ub ← radius
11 return (P_best, radius_best)
```

A. Upper Bound

Circle center c_i has to be closer to p_i than to any other point and line segment on P; otherwise, the circle would intersect P. The valuable tool for determining possible positions of circles' centers satisfying this condition is a Voronoi diagram (VD): the VD for a set of geometries (points and line segments in our case) is a partition of the plane into cells such that each cell contains exactly one input geometry and all points in the plane are closer to the geometry than to any other geometry. $VD(P)$, a Voronoi diagram of points and line segments can be computed by Fortune's sweepline algorithm in $O(n \log(n))$ time and use $O(n)$ space [11].

Fig. 3a shows the VD of a path from Fig. 2. Boundaries of Voronoi cells are formed by points equidistant from two or more input geometries. Boundary curves between two points or two segments are segments, while edges between a point and a segment are parabolic arcs. The center of the largest circle κ_i thus lies on a boundary of i^{th} cell, i.e. the cell VC_i containing i^{th} geometry. Moreover, maximal distances on boundary curves are reached at their end vertices [7].



(a) (b)

Fig. 2: Example solutions of (a) the PCCP, and (b) the WPCCP. The paths (in blue) start at the red points.

Given $VC = \{VC_i\}_{i \in I}$ a set of Voronoi cells containing points on path P we can thus determine upper bound ub of (W)PCCP as:

$$ub = \min_{i \in I} \max_{v \in \Delta(VC_i)} |vp_i|, \tag{1}$$

where $\Delta(VC_i)$ is a set of VC_i's vertices and $|\cdot|$ is the Euclidean distance. In other words, the radius of the largest circle touching a point is the distance of the point to the most distant vertex on its Voronoi cell boundary (see Fig. 3b for an example). The upper bound is the smallest of these radii.

By contradiction, we prove there is no valid solution with a radius larger than ub. Assume p_m for which $|p_m c_m| = ub$. c_m is thus the vertex maximizing the distance in Eq. 1, and m is the cell index for which this maximum is minimal. The solution for radius $r > ub$ contains circle $C_m(\bar{c}_m, r)$ associated with p_m. As c_m is the farthest point of VC_m and $\bar{c}_m$ is farther from p_m than c_m, $\bar{c}_m$ lies outside VC_m. There is, therefore, a point or segment on the path closer to $\bar{c}_m$ than to c_m, i.e., circle $\kappa_m(\bar{c}_m, r)$ has a nonempty intersection with the path. The solution is thus invalid, which is a contradiction.

B. Lower Bound

We determine lower bound lb by finding *some* solution. Assume that a circle center for each cell VC_i lies on a ray α_i which bisects the angle formed by p_i and two edges of VC_i's boundary incident to p_i as shown in Fig. 3c. For each pair $p_i, p_j \in P, i \neq j$ we determine circles' centers c_i, c_j such that:

(1) the centers lie on the bisecting rays: $c_i \in \alpha_i$ and $c_j \in \alpha_j$,
(2) the points lie on the circles with the same radius r_{ij}: $|p_i c_i| = |p_j c_j| = r_{ij}$,
(3) the circles touch: $|c_i c_j| = 2r_{ij}$.

The above conditions lead to a quadratic equation for r_{ij} with one positive solution. Nevertheless, one or both centers can lie outside the individual cells, making the solution invalid. The valid radius $\bar{r}_{ij}$ is thus limited:

$$\bar{r}_{ij} = \min\{r_{ij}, |p_i x_i|, |p_j x_j|\}, \tag{2}$$

where x_i is the intersections of ray α_i with $\delta(VC_i)$, the boundary of VC_i). Similarly, $x_j = \alpha_j \cap \delta(VC_j)$.

The lower bound is the smallest valid radius of all pairs:

$$lb = \min_{i \in I, j \in I, i \neq j} \bar{r}_{ij}.$$

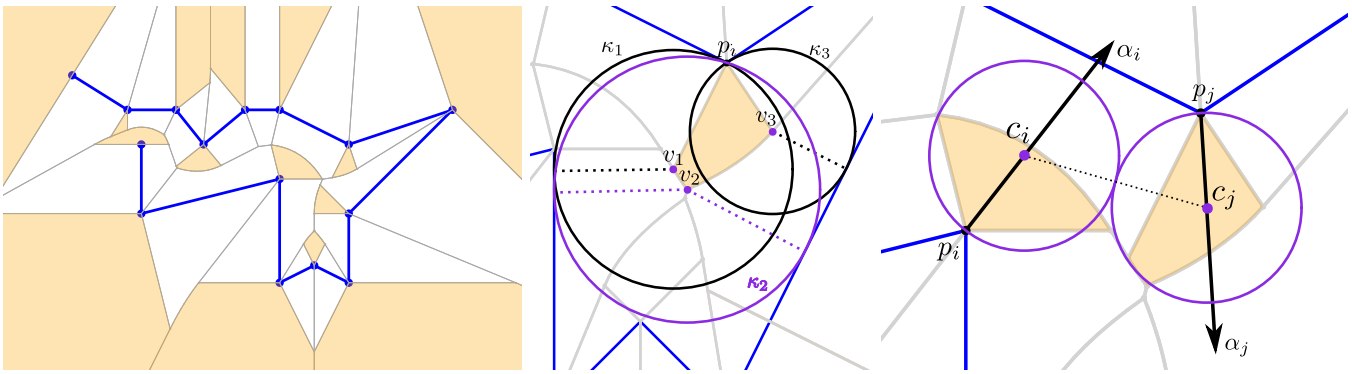
Circle center c_i is computed as the intersection of α_i with a circle centered in p_i with radius lb.

To prove the validity of the solution constructed by this procedure, we must ensure that the constraints C1–C4 from the problem formulation are satisfied. C1 and C2 are met trivially, and C4 holds as c_i lies in VC_i due to Eq. 2. C3 is proved by contradiction using the observation that given two circles touching the same point, a circle with a smaller radius is entirely inside a circle with a larger radius. This means that if the smaller circle intersects with another circle, the larger circle also intersects with the same circle. Assume now that two circles $\kappa_i(lb)$ and $\kappa_j(lb)$ with radii lb associated to some points c_i and c_j intersect. According to the above observation, $\kappa_i(lb)$ and $\kappa_j(\bar{r}_{ij})$ thus intersect as well as $\kappa_i(r_{ij})$ and $\kappa_j(r_{ij})$. This is not possible as $|c_i c_j| = 2r_{ij}$.

C. Placement for a fixed radius

In this section, we describe the algorithm which finds a valid placement of circles for given radius r or reports that such placement does not exist. We formulate this problem as a discrete optimization problem and solve it by a local search heuristic – an initially generated solution is iteratively improved by local optimization. Realize that validity of the initial solution is not guaranteed; thus, the search is done in the space of *all* solutions, not only valid ones. Invalid solutions are penalized in the designed objective function forcing the solver to find a valid one if it exists.

The algorithm shown in Alg. 2 starts by generating a set of valid circles' centers for each Voronoi cell of a point on path P (lines 1–2). Given cell VC_i, the set equals the intersection of the cell and a circle with radius r centered in p_i. To determine the intersection efficiently, the cell is split into sectors according to boundary edges as shown in Fig. 4a. The sectors are processed sequentially in clockwise order. Each sector is described by two angles – directions of rays starting



(a) (b) (c)

Fig. 3: (a) Voronoi diagram (in gray) of the blue path with highlighted cells of points (in orange). (b) Upper bound computation: Voronoi cell of p_i bounded by closed curve $p_1v_1v_2v_3p_1$, circles κ_1, κ_2, and κ_3 going through p_1 with centres v_1, v_2, and v_3, with κ_2 the largest one. The dotted lines connect the circles' centers with the points where the circles touch the path. (c) Lower bound computation: α_1 and α_2 are axes of cells associated with p_i and p_j respectively, s_i and s_j are centers of largest touching circles.

in p_i and passing through endpoints of the boundary edge e. A sequence of points on circle $\kappa(p_i, r)$ lying in the sector is generated for each sector. A point from the sequence lies in VC_i iff it is closer to p_i than to g, where g is a geometry (point or edge) that shares the boundary edge e with p_i.

The initial solution is generated next (line 3) by selecting one circle center from each $\mathcal{V}_i$. Preliminary experiments show that the selection does not influence the solution quality; we thus simply select the center randomly.

The iterative improvement is made in the loop in lines 4-11. In each iteration, points on the path are processed in a random order (lines 8-9). The randomness of the order is ensured by shuffling the indexes (line 7). When processing i^{th} point, all circles' centers are fixed except the i^{th} one and new c_i is selected from $\mathcal{V}_i$ that minimizes the designed objective function (line 9). It consists of two parts.

The first part penalizes candidate centers of circles having a non-empty intersection with other circles:

$$f^i_{int}(c) = \sum_{k \in I \setminus \{i\}} (\text{Area}(\kappa(c, r) \cap \kappa(c_k, r)) + \varepsilon_k), \qquad (3)$$

where

$$\varepsilon_k = \begin{cases} \varepsilon & \text{if } \kappa(c, r) \cap \kappa(c_k, r) \neq \emptyset \\ 0 & \text{otherwise,} \end{cases}$$

ε is a constant (see its meaning bellow), and Area is the area of circles' intersection.

Assume the optimal placement according to f_{int} in Fig. 4b for motivation of the second part. The intersection of κ_1 and κ_2 is small but nonempty. Center c_1 of circle κ_1 is the most left possible, i.e., in the best position. As the intersection of κ_2 and κ_3 is empty, $f^i_{int}(c_3) = 0$ and moving c_3 farther from κ_2 does not improve f^i_{int}. Moving c_2 farther from κ_1 shrinks κ_1–κ_2 intersection, but enlarges the intersection of κ_2 and κ_3 increasing f^i_{int} in total (notice that c_2 is in the optimal position). The solution is to move c_3 to provide space for moving c_2.

The second part of the objective function thus aims to penalize a circle (a center) with circles in its close vicinity even if it does not intersect them:

$$f^i_{dist}(c) = \sum_{k \in N} \gamma(\mu r - |cc_k|),$$

where $N = \{k | k \in I, k \neq i, |cc_k| \leq \mu r\}$, μ and γ are constants. μ specifies the size of the vicinity as the multiple of r and ensures that f^i_{dist} is non-negative. Even a tiny intersection of circles should be penalized more than many non-intersecting circles close to other circles. f^i_{int} should thus always dominate over f^i_{dist} which is the purpose of γ and ε from Eq. 3. We set $\mu = 2.2$, $\varepsilon = 10^{-5}$, and $\gamma = 10^{-10}$.

The new position of c_i minimizes the sum of the two parts:

$$c_i = \min_{c \in \mathcal{V}_i} (f^i_{int}(c) + f^i_{dist}(c)) \qquad (4)$$

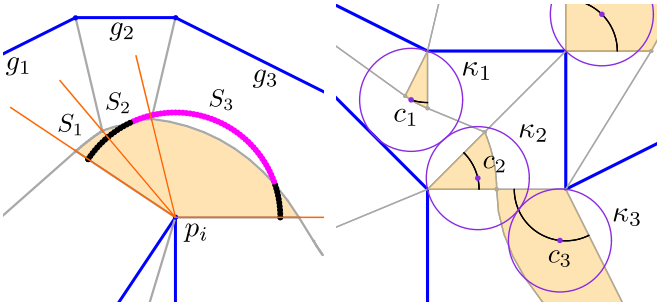
The validity of the solution is determined during the evaluation of f^i_{int}. Simply, the solution is valid if all intersections in Eq. 3 are empty.

We monitor whether the position of some center changed (line 10). If there is no such change during the processing of the entire path, the iterative process finishes (line 11), and the algorithm outputs the result (line 12).

Unfortunately, the cost function in Eq. 4 is not convex, i.e., it can have many local minima. It is thus not guaranteed that Alg. 2 finds a global optimum or even a valid solution if it exists. On the other hand, the algorithm is relatively fast and stochastic. Therefore, we run Alg. 2 several times to increase the probability of a correct result.

D. Modifications for the WPCCP

The presented algorithm is the same for both PCCP and WPCCP, with one exception. The difference lies in the way VC_i , an area of possible positions of circles' centers, is determined for a given point p_i. While a Voronoi diagram (VD) of points and line segments forming the path is computed for the PCCP, and the set of possible centers for a



(a) (b)

Fig. 4: (a) Determination of valid circle centers for a given radius. The orange lines delimit three sections S_1, S_2, and S_3 of point p_i. The black points are valid centers while the magenta ones are invalid. A center from S_k is valid iff it is closer to p_i than to geometry g_k. (b) Situation where the optimization of f_{int} does not find a valid solution.

Algorithm 2: Local search algorithm for a fixed radius.

Input: VC – set of Voronoi cells of points on path P
r – radius
Output: $\langle valid, \Upsilon \rangle$, where $\Upsilon = \{c_i\}_{i \in I}$
$valid$ – flag whether the solution is valid
$\Upsilon = \{c_i\}_{i \in I}$ – set of circles' centers

foreach $i \in I$ **do**
  $\mathcal{V}_i \leftarrow VC_i \cap C(p_i, r)$
  $c_i \leftarrow$ random($\mathcal{V}_i$)
repeat
  $modified \leftarrow false$
  $valid \leftarrow true$
  shuffle(I)
  foreach $i \in I$ **do**
    $\langle valid_i, c_i \rangle \leftarrow$ optimal_center($\mathcal{V}_i$)
    $valid \leftarrow valid \wedge valid_i$
    $modified \leftarrow modified \vee$ changed(c_i)
until $!modified$
return $\langle valid, \Upsilon \rangle$

point is directly its Voronoi cell, the process for the WPCCP is more complex. The placement of a circle for p_i in the WPCCP is influenced only by points and segments not yet visited. This means that VC_i is a Voronoi cell of p_i in the VD constructed for unvisited points and segments. The naïve approach to determine VC_i for all points is to construct the VD of relevant geometries for each point and take its Voronoi cell. The time complexity of this approach is $O(n^2 \log n)$ as a single VD is constructed in $O(n \log n)$ time. Allen et al. [12] propose the VD construction algorithm that incrementally adds new geometries for which the VD is constructed. The amortized complexity of one such insertion is $O(\sqrt{n})$, and thus total complexity of constructing all VC_i's is $O(n\sqrt{n})$. However, we use the naïve approach in our implementation.

IV. EXPERIMENTAL RESULTS

We evaluated the method's performance for both problems on two datasets we created for this purpose. The first dataset consists of 10 instances where the points are distributed evenly. Specifically, we generated hexagonal grids of various sizes, took vertices of these grids as cities, and solved the Travelling Salesman Problem for these cities by the Concorde solver [13]. The TSP solutions specify the instances `hexaXXX`, where `XXX` is the number of points in the instance.

The second dataset (instances `meshXXX`) is generated similarly, but the points are generated as vertices of a conforming constrained Delaunay triangulation (CCDT) generated by [14]. The CCDT distributes points in the plane randomly but tries to keep some minimal distance between them. Examples of both datasets are shown in Fig. 5 together with their solutions.

All experiments were performed within the same computational environment: a notebook with the Intel®Core i5-8250U CPU@1.6 Ghz. The algorithm has been implemented in C++. Twenty runs were run for each instance to provide statistically significant results.

The results are presented in Table I. Each row summarized values of 20 runs for each instance for both problems. lb and up are the lower and upper bounds, r, min, max stand for average, minimal, and maximal found valid radius, σ is the standard deviation of the radius, and $time$ is computational time in seconds.

Several observations can be made from the table. First, the mean radii are closer to the lower bounds than to the upper bounds. If we set $lb = 0\%$ and $ub = 100\%$, the radius is 10-35% for the PCCP, and 13-37% for the WPCCP. This suggests that lb is a better estimate of the optimal radius than ub.

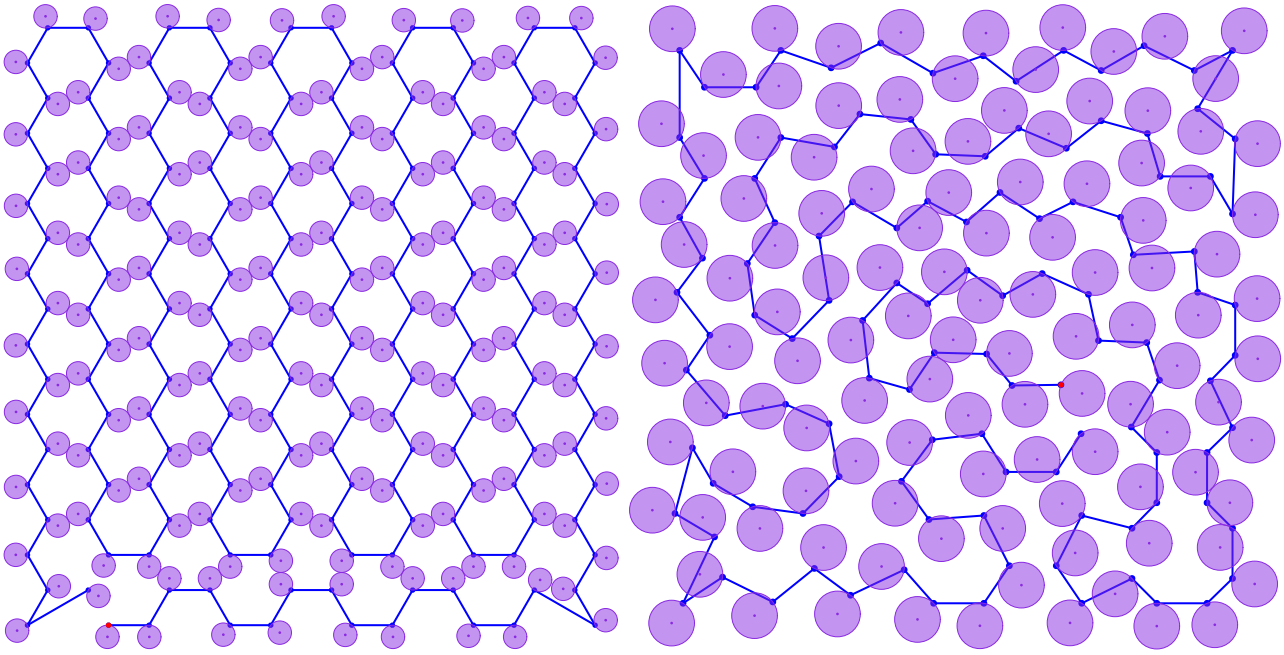
Second, the computational time is two orders higher for the WPCCP. This is caused mainly by the fact that Voronoi cells are larger than in the PCCP, and thus more centers have to be evaluated in Alg.2. Moreover, the convergence of this algorithm is slower due to a higher number of centers.

Although we run Alg. 2 twenty times, the solutions found by the solver differ as the values of σ show. The values are lower for the PCCP, meaning this problem is simpler to solve. Nevertheless, the deviations are relatively low, even for the WPCCP.

V. CONCLUSIONS

We study two problems inspired by an open-pit mining scenario. Contrary to the current approaches, we address the problems of where to place piles that do not obstruct a planned trajectory. Together with a heuristic solver for the problems, we provide upper and lower bounds for a given instance. The experimental evaluation shows that the solver finds good solutions for instances of hundreds of piles/circles in a reasonable time.

While the computational time for the PCCP is sufficient for real deployment, there is space for improvement for the WPCCP. We will thus focus on a more efficient generation of



(a) (b)

Fig. 5: Example instances and theirs solutions. (a) PCCP solution of `hexa180` (b) WPCCP solution of `mesh115`.

	PCCP							WPCCP						
problem	lb	ub	r	min	max	σ	time	lb	ub	r	min	max	σ	time
hexa180	15.00	30.00	17.31	17.31	17.31	0.00	0.50	18.02	103.90	30.66	28.75	31.39	0.64	48.82
hexa240	13.68	27.56	15.62	15.62	15.62	0.00	0.65	16.78	55.14	28.20	26.37	28.77	0.55	60.28
hexa308	12.43	25.00	14.35	14.35	14.35	0.00	0.85	15.02	86.45	24.88	24.14	25.65	0.41	99.93
hexa336	11.75	23.32	13.48	13.47	13.51	0.02	0.93	14.22	80.17	23.73	22.43	24.28	0.47	115.93
hexa416	10.64	21.00	12.23	12.23	12.23	0.00	1.14	13.05	57.95	21.32	20.50	22.00	0.46	134.56
hexa448	9.50	18.75	10.99	10.99	10.99	0.00	1.16	11.52	64.63	19.06	18.12	19.84	0.46	172.08
hexa540	9.50	18.99	10.96	10.96	10.96	0.00	1.50	11.43	65.77	18.63	18.00	19.50	0.41	213.87
hexa576	8.62	17.36	9.90	9.90	9.91	0.00	1.55	10.57	35.34	17.24	16.37	17.82	0.44	196.54
hexa836	7.50	15.00	8.66	8.66	8.66	0.00	2.16	9.04	40.41	14.76	13.93	15.41	0.41	338.74
hexa1144	6.48	13.00	7.37	7.37	7.37	0.00	3.05	7.84	34.95	12.43	11.96	12.92	0.27	413.71
mesh115	14.90	31.04	18.68	18.68	18.68	0.00	0.27	20.72	65.84	34.01	31.75	35.55	1.34	16.82
mesh244	9.25	19.23	10.47	10.29	10.72	0.14	0.37	12.17	38.87	20.22	19.57	20.90	0.49	35.75
mesh268	8.61	19.14	9.95	9.95	9.95	0.00	0.45	11.77	46.47	19.66	17.69	20.62	0.98	54.06
mesh293	7.87	17.87	8.87	8.75	8.96	0.08	0.45	10.98	38.87	17.55	17.07	17.84	0.22	41.52
mesh343	7.78	16.00	8.99	8.94	9.00	0.02	0.42	11.16	45.10	17.40	16.45	18.20	0.56	64.43
mesh374	6.99	14.15	8.76	8.65	8.85	0.06	0.47	9.49	31.50	17.40	16.32	18.11	0.42	66.10
mesh400	7.51	15.98	8.70	8.70	8.70	0.00	0.64	9.72	34.28	15.40	14.70	15.72	0.33	65.52
mesh449	7.06	11.78	8.68	8.66	8.70	0.01	0.45	9.00	26.10	15.32	14.86	15.61	0.18	87.37
mesh686	5.49	12.02	6.47	6.46	6.50	0.01	1.05	6.78	25.22	11.35	11.09	11.82	0.17	112.87
mesh1337	3.51	6.77	4.21	4.20	4.22	0.01	1.21	4.71	17.72	7.85	7.55	8.03	0.12	215.91

TABLE I: Experimental evaluation.

candidate centers in the future. Instead of generating them equidistantly, new candidates will be generated adaptively at promising positions based on the cost value of already evaluated centers.

As Voronoi diagrams can be generated for general geometries, a natural extension of the problem is to consider trajectories of other shapes. An interesting example is Dubin's car, for which optimal paths consist of straight lines and circular turns.

ACKNOWLEDGMENT

The research was supported by Czech Science Foundation Grant No. 23-05104S. The work of David Woller has also been supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS23/122/
OHK3/2T/13. Masoumeh Mansouri is a UK participant in Horizon Europe Project CONVINCE, and her work is supported by UKRI grant number 10042096.

REFERENCES

[1] M. Mansouri, H. Andreasson, and F. Pecora, "Hybrid reasoning for multi-robot drill planning in open-pit mines," *Acta Polytechnica*,

vol. 56, no. 1, pp. 47–56, 2016.

[2] M. Mansouri, F. Lagriffoul, and F. Pecora, "Multi vehicle routing with nonholonomic constraints and dense dynamic obstacles," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3522–3529.

[3] S. Carmesin, D. Woller, D. Parker, M. Kulich, and M. Mansouri, "The hamiltonian cycle and travelling salesperson problems with traversal-dependent edge deletion," *SSRN Electronic Journal*, 2023. [Online]. Available: https://ssrn.com/abstract=4410404

[4] A. Ullrich, J. Hertzberg, and S. Stiene, "Ros-based path planning and machine control for an autonomous sugar beet harvester," in *Proceedings of International Conference on Machine Control & Guidance, (MCG-2014)*, 2014.

[5] M. I. Shamos, "Computational geometry." Ph.D. dissertation, Yale University, 1978.

[6] G. T. Toussaint, "Computing largest empty circles with location constraints," *International Journal of Computer & Information Sciences*, vol. 12, pp. 347–358, 10 1983. [Online]. Available: https://link.springer.com/article/10.1007/BF01008046

[7] M. Schuster, "The largest empty circle problem," in *Proceedings of the Class of 2008 Senior Conference*, 2008, pp. 28—-37.

[8] J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, and S. Sarvattomananda, "Localized geometric query problems," *Computational Geometry*, vol. 46, no. 3, pp. 340–357, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772112001198

[9] J. Augustine, B. Putnam, and S. Roy, "Largest empty circle centered on a query line," *Journal of Discrete Algorithms*, vol. 8, no. 2, pp. 143–153, 2010, selected papers from the 3rd Algorithms and Complexity in Durham Workshop ACiD 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570866709000847

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Numerical recipes: The art of scientific computing," in *Finding Roots of Equations*. Cambridge University Press, 1992, ch. 9.2, pp. 352–355.

[11] S. Fortune, "A sweepline algorithm for voronoi diagrams," in *Proceedings of the Second Annual Symposium on Computational Geometry*, ser. SCG '86. New York, NY, USA: Association for Computing Machinery, 1986, p. 313–322. [Online]. Available: https://doi.org/10.1145/10515.10549

[12] S. R. Allen, L. Barba, J. Iacono, and S. Langerman, "Incremental Voronoi Diagrams," *Discrete & Computational Geometry*, vol. 58, no. 4, pp. 822–848, 2017. [Online]. Available: https://doi.org/10.1007/s00454-017-9943-2

[13] D. Applegate, R. Bixby, V. Chvatal, and W. J. Cook, "The Concorde tsp solver," http://www.math.uwaterloo.ca/tsp/concorde.html, 2003, accessed: 2023-05-01.

[14] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational Geometry*, vol. 22, no. 1, pp. 21–74, 2002, 16th ACM Symposium on Computational Geometry. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772101000475