An Improved Algorithm for the Automated Design of Large-Scaled Robot Skin

Xiangzhi Wei, Ajay Joneja, and Kai Tang

***Abstract*—A recent paper titled "On the Problem of the Automated Design of Large-Scaled Robot Skin" (Anghinolfi *et al.*, 2013) published in the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING addressed the problem of covering the surface of a humanoid robot with the largest number of nonoverlapping equilateral triangular sensor modules. The problem is eventually approximated by a simpler one: how to find the placement of a given polygon P on an equilateral triangular grid G that contains the largest number of the grid triangles. In this paper, we show how to improve the efficiency of the algorithm presented in that paper. Further, we show that the general problem of filling P with the largest number of disjoint equilateral triangles (all entirely contained in P and all of the same size) is not equivalent to that of finding an optimal placement of P on G. Using this result, we propose an improved heuristic for the original problem of covering the skin of a robot with the largest number of triangular sensor modules.**

***Note to Practitioners*—The problem of fitting the largest number of nonoverlapping triangles of an isometric triangular grid in the interior of a simple polygon is very useful, e.g., for computing how to cover the skin of a robot with triangular electronic sensors. We show that the information of the number of triangles that are contained in the interior of a polygon in a given placement can be obtained efficiently by our proposed algorithm. This is a significant improvement over the algorithm provided in Anghinolfi *et al.*, 2013. Furthermore, each connected set of contained triangles can be reported in time linear to the size of the set. Our proposed algorithm is simple and practical, and it can be adapted to efficiently compute the number of cells that are contained in a grid with a tiling of regular polygons.**

***Index Terms*—Automated design, bin-packing, polygon containment, robot skin.**

I. INTRODUCTION

In a recent paper [1], the problem of covering the skin of a robot with the largest number of triangular electronic modules was discussed. This problem is related to another widely studied problem in computational geometry, that of approximating a given surface with an isotropic mesh of triangles [2]–[4] or hexagons [5], [6]. For an introduction to the problem of covering a robot with triangular modules, we refer the reader to [1, Fig. 2] which illustrates the electronic triangular modules. Two underlying assumptions of their approach are: 1) all modules are identical sized equilateral triangles and 2) it is beneficial in practice for most of the triangular modules to share one or more edges.

Manuscript received March 20, 2014; accepted June 12, 2014. Date of publication July 08, 2014; date of current version December 31, 2014. This paper was recommended for publication by Associate Editor J. Gao and Editor K. Lynch upon evaluation of the reviewers' comments. This work was supported in part by UGC GRF under Grant 613312.

X. Wei and A. Joneja are with the Department of Industrial Engineering and Logistics Management, Hong Kong University of Science and Technology, Hong Kong (e-mail: xiangzhi.science@gmail.com; joneja@ust.hk).

K. Tang is with the Department of Mechanical and Aerospace Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. (e-mail: mektang@ust.hk).

Digital Object Identifier 10.1109/TASE.2014.2331692

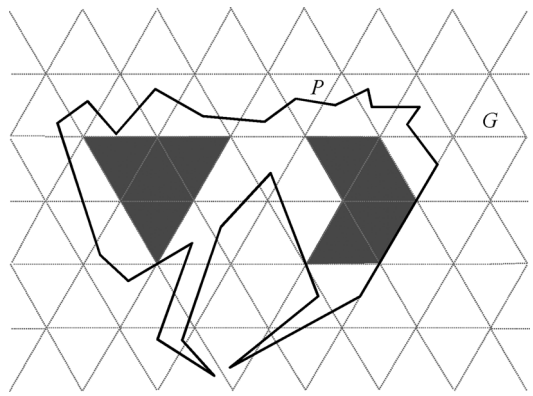

Fig. 1. Placement of a polygon P on a grid G (whose triangles are shown by light gray lines). The triangles that are completely contained in the closure of P are shown in dark gray.

The approach proposed by [1] goes through three stages: 1) *Body part surface unfolding*: unfolding each patch of the surface of the robot (skin) onto a plane; each such patch is approximated by a polygon, P; 2) *Optimal body part coverage*: determining a *placement* (i.e., translation and rotation) of each such polygon that encloses the largest number of triangles in a grid, G, of modules made up of equilateral triangles; and 3) *Optimal bus routing in patches*: defining bus lines and cable routing to connect the triangular modules entirely contained in P. The bulk of their paper focuses on stage (2), i.e., given a simple polygon P and an isometric triangular grid G, find a placement of P on G that maximizes the number of grid triangles contained in P.

Seven heuristics, ($R = \{R_1, R_2, \ldots, R_7\}$), were proposed in [1]. Each of these produces a set of placements of P. Each such placement is evaluated by an algorithm, $\mathbf{EvaluateC}(.)$, which reports the number of triangles contained in P. The main algorithm, $\mathbf{DetermineP}^*(.)$, iteratively uses the heuristics and evaluates placements; the best solution found is reported. We briefly present the idea of $\mathbf{EvaluateC}(.)$ since we shall attempt to improve upon it in this paper. For simplicity, we shall use the same terminology as in [1]. The input to $\mathbf{EvaluateC}(.)$ is a grid G of equilateral triangles and a given placement of a simple polygon P. See Fig. 1 for an illustration of P and G. Each *generating line* of G subtends an angle of 0°, 60°, or 120° w.r.t. the positive x-axis (assumed, without loss of generality, to be horizontal). The algorithm $\mathbf{EvaluateC}(.)$ intersects each generating line with P to obtain a set of intervals that lie inside the closure of P. A counter ns_j stores the number of edges of a triangle c_j, contained on some interval—thus, c_j belongs to an output patch if $ns_j = 3$.

As analyzed in [1], algorithm $\mathbf{EvaluateC}(.)$ runs in $\mathrm{O}(n \times |E_P| \times |MS^*|^2)$ time, where n is the number of generating lines in G, $|E_P|$ is the number of edges on the boundary of P, and $|MS^*| = \arg\max_l\{|MS_l|\}$. We remark that, since $|MS^*| = O(n^2)$, $\mathrm{O}(n \times |E_P| \times |MS^*|^2) = O(n^5 \times |E_P|)$.

The remainder of this paper is organized as follows. In Section II, we propose an algorithm that is more efficient than $\mathbf{EvaluateC}(.)$. In Section III, we propose a new heuristic for the general problem of

1545-5955 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

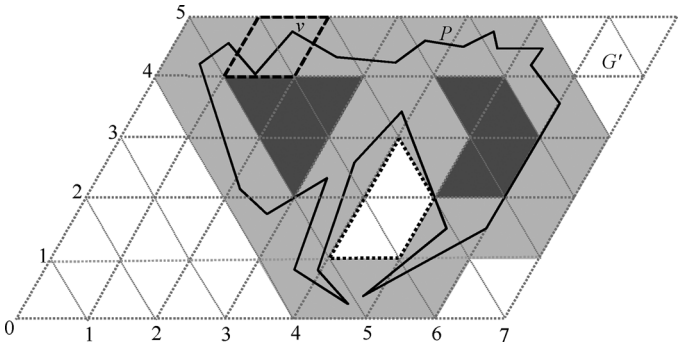

Fig. 2. Placement of a polygon P on a grid G' (whose generating lines are shown by thick dotted lines). The quad [indexed (1, 4)] containing a vertex v of P is shown with dashed border. The holes that are contained in the closure of P is shown in dark gray, and the boundary of the hole that is outside P is shown in dotted lines.

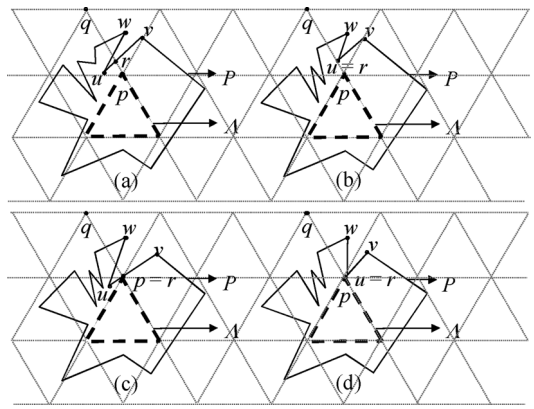

Fig. 3. Determining $\mathrm{vis}(p)$ for a vertex p on a boundary loop Λ of C. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4.

filling an arbitrary polygon with the largest number of disjoint equilateral triangles. Section IV concludes the paper with a summary and a brief discussion.

II. Improved Evaluation Algorithm

As indicated above, the worst-case time complexity of $\mathbf{EvaluateC}(.)$ can be rather poor and therefore affects the performance of the overall search. In this section we provide an improved version of the algorithm to determine the number of triangles of G that are enclosed by a given placement of P.

We shall assume that G is stored in a doubly connected edge list (DCEL) data structure [7]. Let N be the number of cells of G in a region sufficiently large to form a bounding box for all of the possible placements of P. Let G' be the subgraph of G consisting of only the 0° and 60° generating lines; G' naturally induces a quad grid (see Fig. 2). An ordering of the generating lines of G' from bottom to top and from left to right can be trivially determined in $\mathrm{O}(N^{0.5})$ time. Let the 0° lines be indexed from bottom to top, and the 60° lines be indexed from left to right. Each quad in G' can be referenced in terms of the indices of its bottom-left corner; thus the quad cell containing v in Fig. 2 is referenced as (1, 4).

We enhance the data structure of G by associating an attribute $\mathrm{color}(t)$ with each triangle t in G. Since G is represented by DCEL, t can be defined as the face associated with each of the three half-edges (by convention, the left-face of the edge of G corresponding to the half-edge), where a half-edge is a directed version of an edge of T [7]. We shall assume that any half-edge has the same color as its corresponding face. We also associate with each vertex p of G two entries, $[\mathrm{dist}(p), \mathrm{vis}(p)]$, where $\mathrm{dist}(p)$ is a distance field initialized as $+\infty$, and $\mathrm{vis}(p)$ stores an edge of P that is visible from p. $\mathrm{dist}(p)$ is useful in the determination of $\mathrm{vis}(p)$ by a method analogous to that used in ray tracing, as described below. p is contained in the interior of P if it is on the left of $\mathrm{vis}(p)$ [8], [9].

With this data structure, we can compute the number of triangles enclosed by P as follows.

Step 1) Identify the quad d in G' that contains a vertex v of P, and use it to determine the triangle c of G that contains v.

Step 2) Starting from c, compute the connected region of triangles C (in G) that are intersected by the boundary of P (these triangles are highlighted in light gray in Fig. 2). We use a function, $\mathbf{ConcurrentWalk}(.)$, to determine C.

Step 3) C may contain zero or more *holes* (in Fig. 2, these are shown in dark gray if they are in P, or with a dotted boundary if they are outside P). We use a function, $\mathbf{InnerHoles}(.)$, to determine the boundary of each hole that is contained in the closure of P.

Step 4) Compute the number of triangles (denoted by F) enclosed by P using the boundary of each hole obtained in Step (3). These triangles can be reported if required.

In the following, we shall develop algorithms for $\mathbf{ConcurrentWalk}(.)$ and $\mathbf{InnerHoles}(.)$. Using these, we shall develop algorithm $\mathbf{NewEvaluateC}(\boldsymbol{G}, \boldsymbol{P})$ to realize the four steps described above.

To determine $\mathrm{vis}(p)$, we focus on an edge (p, q) of G that is intersected by an edge (v, u) of P. There may be several edges of P that intersect with (p, q), as in Fig. 3(a). In such cases, $\mathrm{vis}(p)$ is selected as (v, u) if it is the edge whose intersection point with (p, q) has the smallest distance from p. Let r denote this intersection point; the value of $\mathrm{dist}(p) = |pr|$ is useful in keeping track of this distance (maintained by lines 11–14 of algorithm $\mathbf{ConcurrentWalk}(.)$ below). Clearly, r is visible from p [8], [9]. We elaborate on how this is implemented in the algorithm. Depending on the position of r with respect to p and u, there are four possible cases (as illustrated in Fig. 3). For convenience of presenting the function $\mathbf{InnerHoles}(.)$, we assume that p is a vertex on a boundary loop Λ of C.

Case 1) r is interior to (v, u), and r does not coincide with p; then $\mathrm{vis}(p)$ is (v, u). See Fig. 3(a) for an illustration.

Case 2) r coincides with u but is distinct from p; then $\mathrm{vis}(p)$ is the edge (of P) incident to u that subtends the smaller non-reflex angle with (u, p). See Fig. 3(b) for an illustration; $\mathrm{vis}(p)$ is the edge (v, u).

Case 3) r coincides with p but is distinct from u; then $\mathrm{vis}(p)$ is (v, u). See Fig. 3(c) for an illustration.

Case 4) r coincides with both p and u; then $\mathrm{vis}(p)$ is (v, u). See Fig. 3(d) for an illustration.

Next we discuss how to determine the boundary of each hole that is contained in P by using the data $[\mathrm{dist}(p), \mathrm{vis}(p)]$ for each vertex p of C. This boundary is a sequence of edges in CCW order around the hole, where each edge is a half-edge of some facet in the DCEL representation of G. Let e_1 be a half-edge directed into p ($p \in C$) that is not gray. Then, the edge in C corresponding to e_1 is on a boundary loop of C. The next edge on this loop in CCW order is obtained as the first non-gray half-edge leaving p (in CCW order from e_1) whose twin half-edge is gray. Let this correspond to the half-edge e_2. Assign a pointer pointing from e_1 to e_2. By repeating this procedure from every newly added half-edge, we can trace the complete loop. Since $\mathrm{O}(1)$ time is required to identify each subsequent edge in the loop, the entire loop is computed in time linear to the number of triangle edges in

its boundary. Let L be the set of boundary loops of C. L can be obtained in $O(g)$ time, where g is the number of triangles in C. We shall assume that this is realized by an algorithm $\mathbf{BoundaryLoops}(\boldsymbol{C})$ whose code is omitted here for brevity. In L there are some loops that are outside P (e.g., the loop whose boundary is indicated by dotted lines in Fig. 2) and those that are inside P (e.g., the boundary loops of the dark regions in Fig. 2). We introduce algorithm $\mathbf{InnerHoles}(\boldsymbol{C})$ to determine the boundaries of the holes that are contained in P. As illustrated in Fig. 3(a)-(b), if a vertex p of a loop Λ satisfies the condition that p is on the left of $\mathrm{vis}(p)$, then Λ is contained inside P. The degenerate cases, where p is on $\mathrm{vis}(p)$ are shown in Fig. 3(c) and (d). Let q be the centroid of a triangle inside Λ with p as one vertex. When p lies in the interior of the edge $\mathrm{vis}(p)$, as in Fig. 3(c), Λ is in P if q is on the left of $\mathrm{vis}(p)$. When p coincides with an endpoint u of $\mathrm{vis}(p)$, we set $\mathrm{vis}(p)$ as the edge of P incident to u that subtends the smaller non-reflex angle with (u, q). In Fig. 3(d), this edge is (v, u). Then Λ is in P if q is on the left of $\mathrm{vis}(p)$.

Algorithm 1: $\mathbf{ConcurrentWalk}(\boldsymbol{G}, \boldsymbol{P}, \boldsymbol{c}, \boldsymbol{v_0})$

Input. Grid G, a given placement of polygon P, a triangle c of G containing a vertex v_0 of P.

Output. A connected region C in G that consists of the triangles intersected by the boundary of P, where each triangle of C is marked as 'gray' and $[\mathrm{dist}, \mathrm{vis}]$ of each vertex of C is updated.

1. $C := \{c\}$, $color(c) :=$'gray', $v := v_0$; // initialization
2. $u :=$ next CCW vertex of P adjacent to v;
3. **if** $u = v_0$ **then return** $\boldsymbol{C}$;
4. **if** $u \in c$ **then**
5. $\quad v := u$;
6. $\quad$ **goto** step 2;
7. **else**
8. $\quad b :=$ the triangle adjacent to c intersected by (v, u);
9. $\quad$ **if** $color(b) \neq$ 'gray'; **then** $color(b) :=$'gray';
10. $\quad (p, q) :=$ the edge of b that is intersected by edge (v, u) at a point r;
11. $\quad$ **if** $\mathrm{dist}(p) > |pr|$ **then**
12. $\quad\quad \mathrm{dist}(p) := |pr|$;
13. $\quad\quad$ **if** ($r = u$ and $r \neq p$) **then** $\mathrm{vis}(p) :=$ the edge (of P) incident to u that subtends the smaller non-reflex angle with (u, p);
14. $\quad\quad$ **else** $\mathrm{vis}(p) := (v, u)$;
15. $\quad$ **if** $\mathrm{dist}(q) > |qr|$ **then**
16. $\quad\quad \mathrm{dist}(q) := |qr|$;
17. $\quad\quad$ **if** ($r = u$ and $r \neq q$) **then** $\mathrm{vis}(q) :=$ the edge (of P) incident to u that subtends the smaller non-reflex angle with (u, q);
18. $\quad\quad$ **else** $\mathrm{vis}(q) := (v, u)$;
19. $\quad$ **if** $color(b) \neq$'gray' **then** $C := C \cup \{b\}$;
20. $\quad c := b$;
21. $\quad$ **if** u is not contained in b **then goto** step 8;
22. $\quad$ **else**
23. $\quad\quad v := u$;
24. $\quad\quad$ **goto** step 2;

In each iteration, function $\mathbf{ConcurrentWalk}(.)$ either steps forward by one edge of P (lines 2–6), or by one triangle of G that is intersected by a edge of P (lines 8–21). In this process, lines 8–21 maintain an edge (v, u) of P that intersects with an edge (p, q) of G and updates $\mathrm{dist}(.)$ and $\mathrm{vis}(.)$ for p and q. The boundary of P may intersect a triangle of G multiple times, as can be seen in the lower portions of P in Fig. 2. Let I be the number of intersections between the triangles of G and the boundary of P. Let l be the length of an edge of a triangle in G, and L_E be the perimeter of P. Let E_S denote the set of all edges of P that are no longer than l, and E_L be the set of all the remaining edges of P; let L_L be the sum of lengths of all edges in E_L. Then:

Theorem 1: $I = O\left(|E_P| + L_L/l\right)$.

Proof: Let l_e be the length of an edge e of P. If $l_e \leq l$, it either stays in the interior of a triangle of G, or it intersects with $O(1)$ triangles (the maximum is when e is coincident with an edge of G, in which case it intersects with 10 triangles). So the total number of intersections between edges of E_S and the edges of G is $O(|E_S|)$.

If $l_e > l$, then we can decompose e into $\lfloor l_e/l \rfloor + 1$ segments, each of which has $O(1)$ intersections with G. Therefore, the number of triangles of G that are crossed by all the long edges is $\Sigma_e(\lfloor l_e/l \rfloor + 1)$, $e \in E_L$. $\Sigma_e(\lfloor l_e/l \rfloor + 1) \leq |E_L| + L_L/l$. Since $|E_L| + |E_S| = |E_P|$, it follows that $I = O(|E_P| + L_L/l)$. □

It follows from the above that the time required to run algorithm $\mathbf{ConcurrentWalk}(.)$ is $O(|E_P| + I)$, which is $O(|E_P| + L_L/l)$, where $|E_P|$ is the number of edges of P. Next we describe the algorithm to identify the holes inside P.

Algorithm 2: $\mathbf{InnerHoles}(\boldsymbol{C})$

Input: C.

Output: $L_{in} = \{L_1, \ldots, L_h\}$, where L_i is the list of triangle edges bounding the i-th hole inside P, and h is the number of holes that are contained inside P.

1. $L :=$ $\mathbf{BoundaryLoops}(\boldsymbol{C})$;
2. **for** each list Λ in L **do**
3. $\quad p :=$ the first vertex in Λ;
4. $\quad$ **if** p is on the left of $\mathrm{vis}(p)$ **then** $L_{in} := L_{in} \cup \{\Lambda\}$;
5. $\quad$ **if** p is on $\mathrm{vis}(p)$ **then**
6. $\quad\quad t :=$ a triangle inside Λ with p as one vertex;
7. $\quad\quad q :=$ centroid of t;
8. $\quad\quad$ **if** p coincides with an endpoint u of $\mathrm{vis}(p)$ **then**
9. $\quad\quad\quad \mathrm{vis}(p) :=$ the edge (of P) incident to u that subtends the smaller non-reflex angle with (u, q);
10. $\quad\quad$ **if** q is on the left of $\mathrm{vis}(p)$ **then** $L_{in} := L_{in} \cup \{\Lambda\}$;
11. **return** L_{in};

Line 1 requires $O(g)$ time, where g is the number of triangles that intersect the boundary of P. The for-loop from line 2 to line 10 is executed $O(|L|)$ times, requiring $O(1)$ time in each iteration. Since $|L| < g$, the total time required is $O(g)$. Using $\mathbf{ConcurrentWalk}(.)$ and

$\mathbf{InnerHoles}(.)$, we introduce algorithm $\mathbf{NewEvaluateC}(\boldsymbol{G}, \boldsymbol{P})$ that improves upon algorithm **EvaluateC**(Algorithm 1 in [1]).

Algorithm 3: $\mathbf{NewEvaluateC}(\boldsymbol{G}, \boldsymbol{P})$

Input. Grid G and a fixed placement of P.

Output. An integer F.

1. $v :=$ any vertex of P; $F := 0$;

2. $d :=$ locate the quad in G' containing v; $c :=$ identify the triangle in d containing v;

3. $C := \mathbf{ConcurrentWalk}(\boldsymbol{G}, \boldsymbol{P}, \boldsymbol{c}, \boldsymbol{v})$;

4. $L_{in} := \mathbf{InnerHoles}(\boldsymbol{C})$;

5. **for** each hole H_i in L_{in} **do**

6. $\quad A_i :=$ area of H_i;

7. $\quad N_i = A_i/a$; // a is the area of a single triangle in G

8. $\quad$ mark H_i with N_i; // this is useful in comparing different placements of P.

9. $\quad F := F + N_i$;

10. **return** F;

Line 1 requires $\mathrm{O}(1)$ time to find a vertex v in P. Line 2 identifies the quad, d, in G' that contains this vertex. The indices of the bottom-left corner of d can be computed as $(\lfloor v_x/l \rfloor, \lfloor v_y/(l \sin \pi/3) \rfloor)$, where (v_x, v_y) are the coordinates of v, and l is the length of an edge of any triangle in G; this takes $\mathrm{O}(1)$ time. Thereafter, c can be determined in $\mathrm{O}(1)$ time since d is made up of two triangles in G. Line 3 requires $\mathrm{O}(|E_P| + I)$ time, and line 4 requires $\mathrm{O}(g)$ time as analyzed above. Let $(v_1, v_2, \ldots, v_m)$ be the vertices of a hole H_i in CCW order, then the area of the hole A_i can be computed as follows:

$$A_i = \frac{1}{2}\left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_m & y_m \\ x_1 & y_1 \end{vmatrix} \right). \quad (1)$$

In line 6, A_i can be computed in $\mathrm{O}(m)$ time. The total number of vertices in all holes is bounded by $\mathrm{O}(g)$. Each of steps 7–9 in the for-loop require $\mathrm{O}(1)$ time; therefore the total time for the for-loop from lines 5–9 is bounded by $\mathrm{O}(g)$. The discussion above can be summarized in the following theorem.

Theorem 2: Given a grid G of N equilateral triangles, G can be preprocessed in $\mathrm{O}(N)$ time such that the number of triangles enclosed by any simple polygon P with $|E_P|$ edges can be determined in $\mathrm{O}(|E_P| + I)$ time, where I is the number of intersections between the triangles of G and the boundary of P. If F is the number of triangles enclosed by P, then these triangles can be reported in additional $\mathrm{O}(F)$ time.

Proof: The running time of the preprocessing algorithm and that of Algorithm $\mathbf{NewEvaluateC}(.)$ was analyzed above. Furthermore, all triangles enclosed in P can be reported in output sensitive $\mathrm{O}(F)$ time by traversing the DCEL data structure [7]. □

Theorem 2 can be applied to the placement of P on any grid G with a tiling of regular polygons.

III. Improved Heuristics for the Polygon Containment Problem

Observe that an optimal placement of P on G cannot always guarantee that P encloses the maximum number of interior-disjoint triangular modules. For example, an optimal placement of P in Fig. 4 encloses four triangles, but P can enclose three additional triangles.

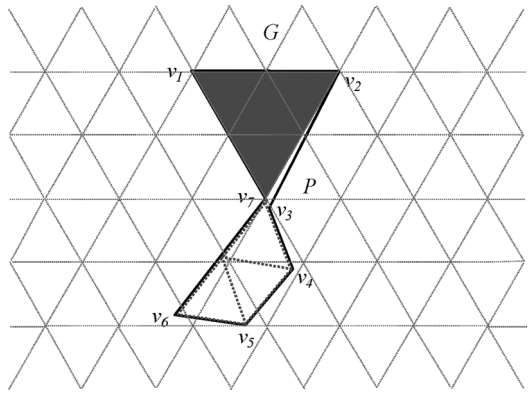

Fig. 4. P can contain three additional triangular modules (shown in dotted lines) in the region enclosed by a polygonal loop $(v_3, v_4, v_5, v_6, v_7, v_2, v_3)$, where each triangular module is equal to a triangle in G (upon a rotation), but this cannot be realized by placing P on G.

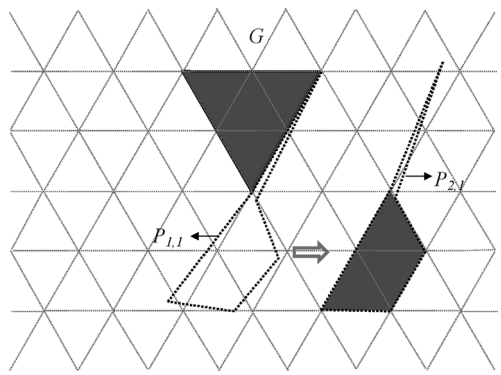

Fig. 5. A new placement of $P_{1,1}$, i.e., $P_{2,1}$, results in an additional containment of three triangles of G.

Packing the largest number of triangles inside a polygon is a special case of the well known bin-packing problem [10].

In this section, we propose a heuristic to further increase the number of triangular modules contained in P by an adaptation of the approach presented in [1]. As can be seen from the example in Fig. 2, there is no practical reason for the two (disconnected) dark gray holes to maintain the same orientation with respect to P. Using this observation, we propose a greedy heuristic to decompose P into a number of subpolygons to potentially increase the number of triangular modules it encloses. Our approach uses the property that any placement of P produced by algorithm $\mathbf{DetermineP}^*$ (Algorithm 2 in [1]) is in contact with one or more holes at a few points.

In each iteration of our algorithm, the largest hole that is in contact with P, denoted by H, is removed from P. The closure of $(P - H)$ is a set of simple polygons, after we split the result at the contacting points (if any) of H and P. Each newly generated subpolygon is associated with the number of triangular modules enclosed in it in the current placement.

For each subpolygon obtained as above, we call algorithm $\mathbf{DetermineP}^*$ to determine its placement on G. If the number of triangular modules enclosed is larger than that in its previous placement, then we accept the new placement of the (reduced) polygon and continue the iterative process. Otherwise we stop at the incumbent placement.

In the remainder of the paper, we shall assume that the call to $\mathbf{EvaluateC}(.)$ in line 8 of algorithm $\mathbf{DetermineP}^*(G, P, R)$ (Algorithm 2 in [1]) has been replaced by a call to our algorithm

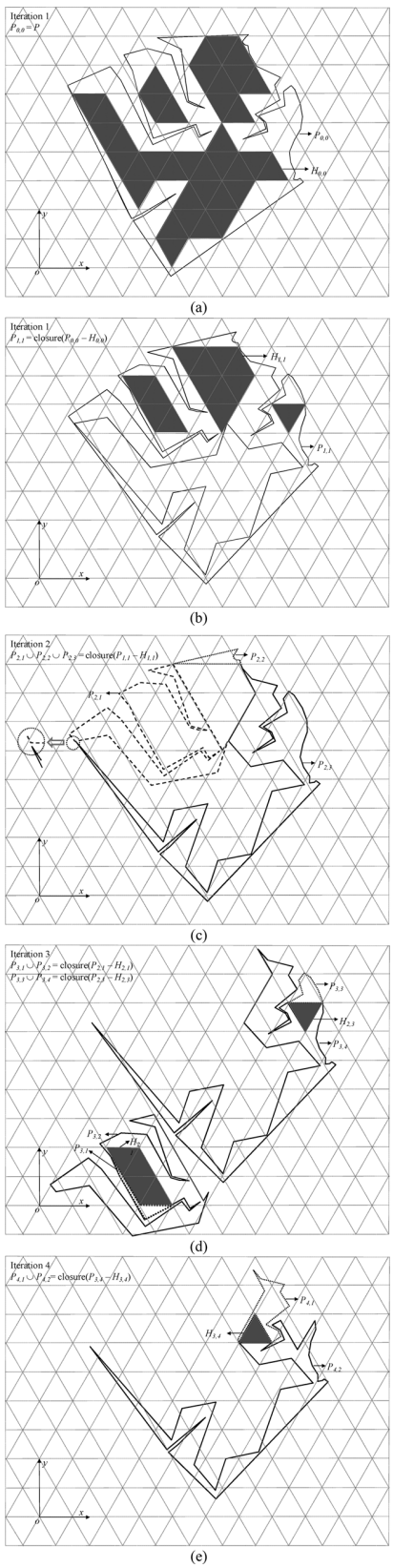

Fig. 6. Illustration of the subpolygons across multiple iterations of running $\mathbf{ImprovedContainment}(\boldsymbol{G}, \boldsymbol{P_{0,0}}, \mathbf{0})$ for the polygon in [1, Fig. 16]. Different polygon boundary styles indicate the interior-disjoint subpolygons after removing the largest hole in each iteration. The grid is fixed in a coordinate frame o–xy. The subpolygons that cannot contain any new hole are eliminated in each subsequent iteration.

$\mathbf{NewEvaluateC}(G, P)$. For ease of description, we denote by $P_{i,j}$ the jth subpolygon in the ith iteration. Thus $P_{0,0}$ denotes the initial input P. Let $N(P_{i,j})$ be the number of triangular modules enclosed in $P_{i,j}$. Algorithm $\mathbf{ImprovedContainment}(.)$ below realizes our proposed scheme, when it is invoked with the initial inputs $(G, P_{0,0}, 0)$.

Algorithm 4: $\mathbf{ImprovedContainment}(\boldsymbol{G}, \boldsymbol{P_{i,j}}, \boldsymbol{i})$

Input. Grid G and a given placement of $P_{i,j}$.

Output. F, the number of triangular modules that can be contained in $P_{i,j}$.

1. **if** $i = 0$ and $j = 0$ **then** $F := 0$; // intialization
2. $H_{i,j} :=$ the largest hole enclosed by and touching $P_{i,j}$;
3. **if** $H_{i,j} = NULL$ **then return** 0;
4. $F := F + N(H_{i,j})$ // $N(H_{i,j})$ is the number of triangles in $H_{i,j}$
5. $S = \{P_{i+1,1}, P_{i+1,2}, \ldots, P_{i+1,s}\} := \text{closure}(P_{i,j} - H_{i,j})$; // after splitting $P_{i,j}$ at each contact point with $H_{i,j}$
6. **for** $x = 1$ to s **do** $\mathbf{DetermineP}^*(G, P_{i+1,x}, \mathrm{R})$;
7. **if** $N(S) \leq N(P_{i,j}) - N(H_{i,j})$ **then return** F;
8: **else**
9. **for** $x = 1$ to s **do**
10. $F := F + \mathbf{ImprovedContainment}(\boldsymbol{G}, \boldsymbol{P_{i+1,x}}, \boldsymbol{i+1})$;

Algorithm $\mathbf{DetermineP}^*(.)$ uses seven heuristics, and therefore we omit the running time analysis of $\mathbf{ImprovedContainment}(.)$, except to mention that in line 5, the time required to remove $H_{i,j}$ is bounded by $\mathrm{O}(|H_{i,j}|)$, where $|H_{i,j}|$ is the number of edges on the boundary of $H_{i,j}$. The above algorithm returns the number of triangular modules by our new placement strategy; if we wish also to report the actual triangles and their location in P, then it can be adapted with minor modifications and using an additional $\mathrm{O}(F)$ time.

Fig. 5 shows an illustration of decomposing the polygon in Fig. 4 and repositioning its subpolygons. Fig. 6 shows the illustrations of processing the placement of P shown in [1, Fig. 16] by using our proposed scheme.

IV. Conclusion

In this paper, we improved the results and approach presented in [1] for the problem of covering a given simple polygonal shape by the largest number of equilateral triangular modules. Our first contribution is an improved evaluation algorithm that reports the number of triangles enclosed by a given placement of a polygon on a triangular grid. Our second contribution is an improved heuristic scheme that can potentially increase the number of triangular modules that may be placed inside the given polygon, while respecting the practical constraints of the application, namely covering the skin of a humanoid robot by sensor modules.

We conclude the paper with a discussion about the difficulty of the current problem. Without imposing any engineering constraints, the problem of packing the largest number of equilateral triangles in a larger equilateral triangle has been studied before in [11] and [12]. In both of those papers, only the non-rotational form of the problem was studied, namely, all the triangles are in edge-parallel positions. However, the combinatorial complexity of even that restricted problem is not discussed, and to the best of our knowledge, the problem is still open. Furthermore, the problem of packing the largest number of

axis-parallel unit squares into a simple polygon is also open (see [13, problem 56]). Therefore we believe that packing equilateral triangles in a simple polygon is also open.

REFERENCES

[1] D. Anghinolfi, G. Cannata, F. Mastrogiovanni, C. Nattero, and M. Paolucci, "On the problem of the automated design of large-scale robot skin," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 4, pp. 1087–1100, Oct. 2013.

[2] P. Alliez, E. C. de Verdiere, O. Devillers, and M. Isenburg, "Isotropic surface remeshing," in *Proc. Shape Modeling Int.*, 2003, pp. 49–58.

[3] M. Singh and S. Schaefer, "Triangle surfaces with discrete equivalence classes," *ACM Trans. Graphics*, vol. 29, no. 4, 2010, Art. ID 46.

[4] Y. Li, E. Zhang, Y. Kobayashi, and P. Wonka, "Editing operations for irregular vertices in triangle meshes," *ACM Trans. Graphics*, vol. 29, no. 6, 2010, Art. ID 153.

[5] E. Biyikli, J. Liu, X. Yang, and A. C. To, "A fast method for generating atomistic models of arbitrarily-shaped carbon graphitic nanostructures," *RSC Advances*, vol. 3, pp. 1359–1362, 2013.

[6] W. Wang, Y. Liu, D. Yan, B. Chan, R. Ling, and F. Sun, "Hexagonal meshes with planar faces," Hong Kong University, Tech. Rep. CS-2008–13, 2008.

[7] M. de Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin, Germany: Springer-Verlag, 2008.

[8] W. P. Horn and D. L. Taylor, "A theorem to determine the spatial containment of a point in a planar polygon," *Comput. Vis., Graphics and Image Process.*, vol. 45, pp. 106–16, 1989.

[9] J. Liu, Y. Q. Chen, J. M. Maisog, and G. Luta, "A new point containment test algorithm based on preprocessing and determining triangles," *Computer-Aided Design*, vol. 42, pp. 1143–1150, 2010.

[10] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-Hard Problems*, D. Hockbaum, Ed. Boston, MA, USA: PWS, 1996, pp. 46–93.

[11] Y. Zhang and Y. Fan, "Packing and covering a unit equilateral triangle with equilateral triangles," *Electron. J. Combinatorics*, vol. 12, p. #R55, 2005.

[12] A. Dumitrescu and M. Jiang, "On a covering problem for equilateral triangles," *Electron. J. Combinatorics*, vol. 15, p. #R37, 2008.

[13] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke, "The Open Problems Project," [Online]. Available: http://maven.smith.edu/~orourke/TOPP