Non-convex polygons clustering algorithm

Alexey Kruglikov[1,*], and *Mikhail* Vasilenko[1]

[1]Ural Federal University, Yekaterinburg, Russia

Abstract. A clustering algorithm is proposed, to be used as a preliminary step in motion planning. It is tightly coupled to the applied problem statement, i.e. uses parameters meaningful only with respect to it. Use of geometrical properties for polygons clustering allows for a better calculation time as opposed to general-purpose algorithms. A special form of map optimized for quick motion planning is constructed as a result.

1 Introduction

A variety of real-world applications require using random maps of complex structure; similar problems arise in computational dynamics [1], blood flow dynamical simulation [2], and motion planning for limited maneuverability objects [3].

Specific kinds of maritime motion planning problems exist where it is important to consider all the obstacles charted on the map. In case of a major archipelago and reasonable level of detail this could mean dealing with thousands of islands described with up to tens of thousands points each, which is costly in terms of computational resources. A motion planning algorithm currently developed in Ural Federal University is trying to reduce costs by working on a pre-clustered map. This paper is concerned with an algorithm of preliminary clustering.

Approach used is similar to hierarchical clustering [4], its specific features are as follows:

- hierarchy has a fixed number of levels;
- different clustering procedures are used for different levels;
- different properties are inherent to clusters of different levels.

2 Problem and terms

Consider a map with charted obstacles, which may be represented as a two-dimensional Cartesian coordinate system and a set of polygons with coordinates measured in it. It is assumed that polygons are identified by unique names or indexes, have no self-intersections, and do not intersect pairwise. Also a positive constant value R_{max} is defined, meaning minimal safe distance from an obstacle the vehicle in question should not violate.

Such obstacles clusters must be found that their convex geometric borders had at least 2 R_{max} distance pairwise, i.e. allowed safe movement between them.

*a.s.kruglikov@urfu.ru

It is convenient to use the following notations:

- *Frame* is a minimal rectangle aligned along coordinate axes and containing a nonempty set of polygons.
- *Container* is a set of polygons with one member marked as main, used as a low-level polygon cluster.
- *Container family* is a nonempty set of containers, mid-level polygon cluster.
- *Families group* is a nonempty set of families, highest-level polygon cluster.
- *Rough hull* is a convex hull of a set of frames, used as a polygon cluster geometrical border.

3 Algorithm

Clustering is done in a "bottom-up" fashion in four steps:

- frames are calculated for the polygons;
- polygons are clustered into containers;
- containers are clustered into families;
- families are clustered into groups.

It is noteworthy that the first two steps can benefit from parallel implementation. They are also discussed in greater detail since are most important and time-consuming.

Frame calculation and container clustering. A frame for a polygon is calculated by brute force; it represents a polygon in the rest of clustering process. In essence, two issues are addressed: all the polygons of original data widely varying in their properties are given unified representation, and also data dimension is reduced. What is lost in precision is compensated for by lower calculation time given typically long and complex polygon boundaries.

Definition. A *container*, notation C, is a set of polygons with one member marked as main such that all polygons' frames are included as geometrical shapes into main polygon frame; the latter is also the frame of the container itself.

A container might not include the frame of another container.

In the following P_i is used to denote a polygon with index i, while its frame is denoted as F_i; P_0 denotes main element, and $F(C)$ is containers frame.

$$C = \left\{ P_i \mid \forall i > 0 : F_i \subset F_0 \right\}$$

$$F(C) = F_0$$

$$C_1 \neq C_2 \rightarrow F(C_1) \not\subset F(C_2) \wedge F(C_2) \not\subset F(C_1) \tag{1}$$

Figure 1 shows polygons and corresponding containers, the latter depicted by their frames. It is convenient to identify a container by its main member name.

Container clustering algorithm operates a dynamic list CL of currently found containers. Any freshly constructed frame representing polygon P is checked for nesting into each current container. There could only be three cases:

- A frame is included into at least one container frame. Then P becomes member to each container it fits into.
- A frame fits into no current container; P becomes a new one-element container in CL.
- A frame includes a frame of at least one current container. A new container with P as main member is put into CL, while included containers are removed, and all their members are transferred to the new container.

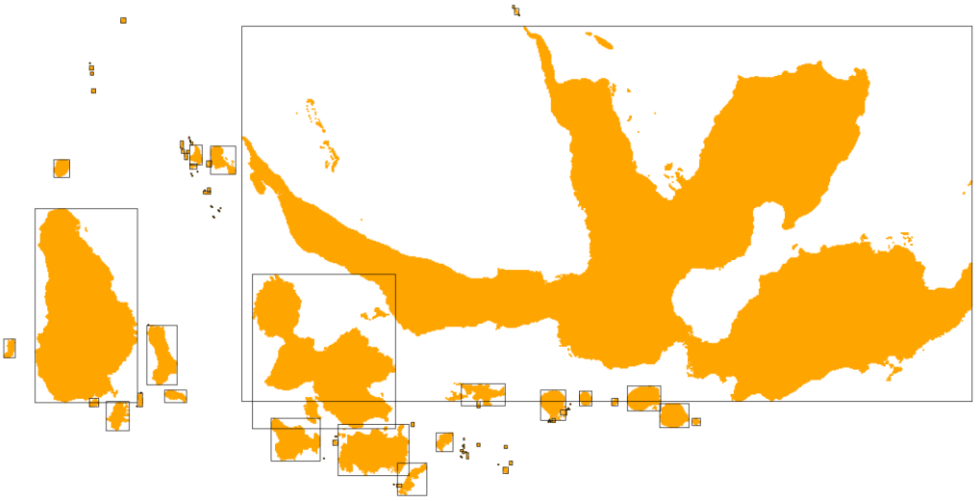

Fig. 1. Containers.

Proposition 1. Container clustering algorithm guarantees that (1) holds.

Proof. Assume $\exists C_1 \neq C_2 : F(C_1) \subseteq F(C_2)$.

If C_1 was constructed later than C_2, then at the moment of its main member P frame analysis C_2 was in the current containers list. Since

$$F(P) = F(C_1) \subseteq F(C_2),$$

P should have been included into C_2 and any other container it fitted into. Therefore, to have P as its main element C_1 should have already existed by the time it was constructed, which is impossible.

If C_2 was constructed later than C_1, then at the moment its main element P frame was analyzed C_1 was in the current container list. Since

$$F(C_1) \subseteq F(C_2) = F(P),$$

C_1 should have been removed from the list at that moment. But C_1 exists, so it must have been added later, which is impossible as was already shown.

qed

It is worth noting that calculation of frame nesting typically also yields information on frames intersection which is useful for the next step and could be stored. For that a list for intersecting containers registration is attached to each container, all the containers intersecting the frame are stored in a list L, and the following logic is used after the frame is checked for nesting:

- If a container includes a frame, then it intersects frames of all containers in L.
- If P becomes a new one-element container, it intersects all containers in L.
- If a frame includes one or more containers, it intersects frames of all containers in L plus all containers the nested ones did intersect, and minus containers included.

At the end of clustering procedure all the containers in CL are checked for their lists of intersections, and the latter are merged into an undirected graph G of intersections. For the later discussion it is important that all the containers intersections are thus found.

Proposition 2. If container clustering algorithm stores intersection information, all container intersections are found.

Proof. Assume $\exists C_1, C_2 : F(C_1) \cap F(C_2) \neq \varnothing$, but G does not contain rib (C_1, C_2).

If C_1 was constructed later, then at the moment of its main element P frame analysis C_2 was in a list of current containers. Then a list L of frame-intersecting containers for P did have C_2 in it. The only chance to miss C_2 in the list of C_1 is to drop it since C_2 was included into C_1, which is not true.

If C_2 was constructed later, the same logic shows C_2 must have C_1 in its list.

Either way graph G obtains rib (C_1, C_2), which contradicts assumption.

qed

Complexity and parallelism. The best case for the algorithm is obviously when the first frame contains all the rest, and it is enough to do one check for each frame at a cost of $O(N)$ operations where N is the number of polygons. The worst case is the opposite, when all the frames but the last are independent though maybe intersecting, while the last is including all of them. At least N^2 operations are needed then.

That said, it is obvious that in a sequential implementation algorithm would benefit from preliminary sorting of frames, and it would be natural to sort frames by their area.

Parallel implementation should consider the following: frame calculation can be much more time-consuming than a step of clustering, so it is possible to link two procedures in a pipeline fashion. Container clustering itself does intersection checking of a frame independently, so it could be easily implemented parallel, but editing the *CL* list prevents it from checking two different frames simultaneously. Also, since short-length polygons will usually get their frames calculated faster, clustering will often start with smaller polygons, which is undesirable. With respect to that a parallel implementation could be as follows.

Frames calculation is done with as much parallelism as possible. Container clustering procedure operates a buffer of incoming frames and starts only when a buffer is full. It then copies buffer contents, clears buffer, sorts copied frames by their area and runs sequentially, maybe using parallelism when checking frame nesting/intersection with different members of *CL* list. While that is done, a buffer is getting new frames for the next round of container clustering.

Family clustering. After all polygons are sorted into containers further clustering is done based on container frames intersection.

Definition. *Container family*, notation *CF*, is a nonempty set of containers such that for each two different members there exists a sequence of frame-intersecting members of the same family connecting them.

Containers from different families might not have intersecting frames.

$$CF = \{C_i\}$$

$$\forall C_i, C_j \in CF \; \exists C_{k_0}, \ldots, C_{k_n} \in CF : C_{k_0} = C_i, C_{k_n} = C_j, F\left(C_{k_l}\right) \cap F\left(C_{k_{l+1}}\right) \neq \varnothing$$

$$CF_1 \neq CF_2 \rightarrow \forall C_1 \in CF_1, C_2 \in CF_2 \; F(C_1) \cap F(C_2) = \varnothing$$

If the previous step did use information on frames intersection, which is preferable, then graph G of intersections is already constructed, and a family is a simply connected component in it. As was shown, all container intersections are found by the previous step, which guarantees correct families construction.

Otherwise graph G is to be calculated in a separate step of $O(N^2)$ complexity where N is number of containers. Family clustering algorithm is a simplified form of single-link clustering [5].

Figure 2 shows containers from Figure 1 clustered into families depicted by red borders.

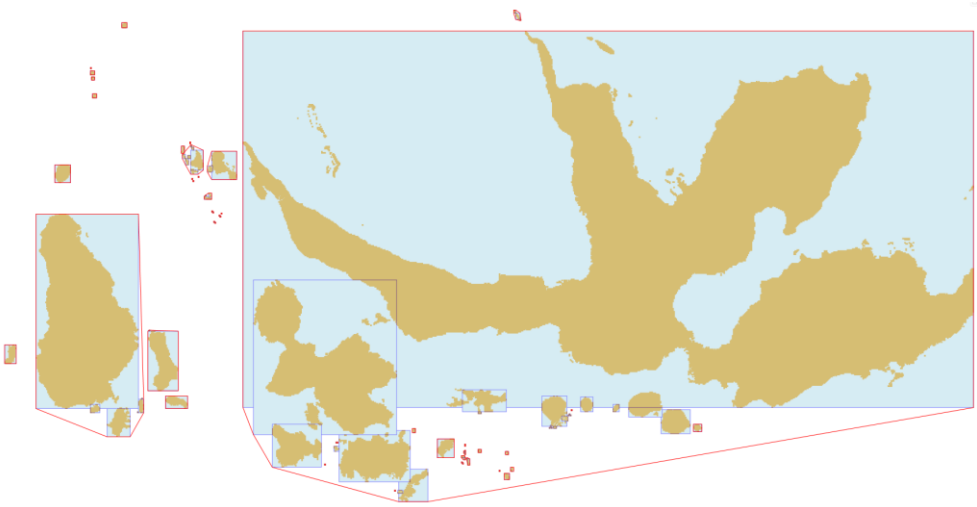

Fig. 2. Container families.

Groups clustering. To cluster families into groups each of the former is assigned a rough hull; it is worth noting that rough hulls of different families might be intersecting or even included as geometrical shapes. Different groups should have their rough hulls no closer than 2 R_{max}; intersection or close distance between them invoke merging. Figure 3 shows several families groups depicted by red borders; families from Figure 2 can be seen in the upper right corner.

Used algorithm is also a modification of single-link clustering, but is done in two steps, and distance is not derived from distances between families but rather recalculated as a distance between groups' rough hulls. First, groups are formed of families with intersecting or nested rough hulls. This step is finished when all the groups have non-intersecting rough hulls. The second step uses R_{max} and merges groups with distances lower than twice that value. Resulting groups have their rough hulls no closer than 2 R_{max} to each other, which is the solution for the stated problem.

Second step is distinguished for the following reason: all the steps before it are only using polygons coordinates and thus produce natural clustering, which could be saved and reused with different values assigned to R_{max}.

Land clusters hierarchy. As a result a four-level land clusters hierarchy is formed with levels of:

- polygons and their respective frames;
- containers and their respective frames;
- families and their respective rough hulls;
- family groups and their respective rough hulls.

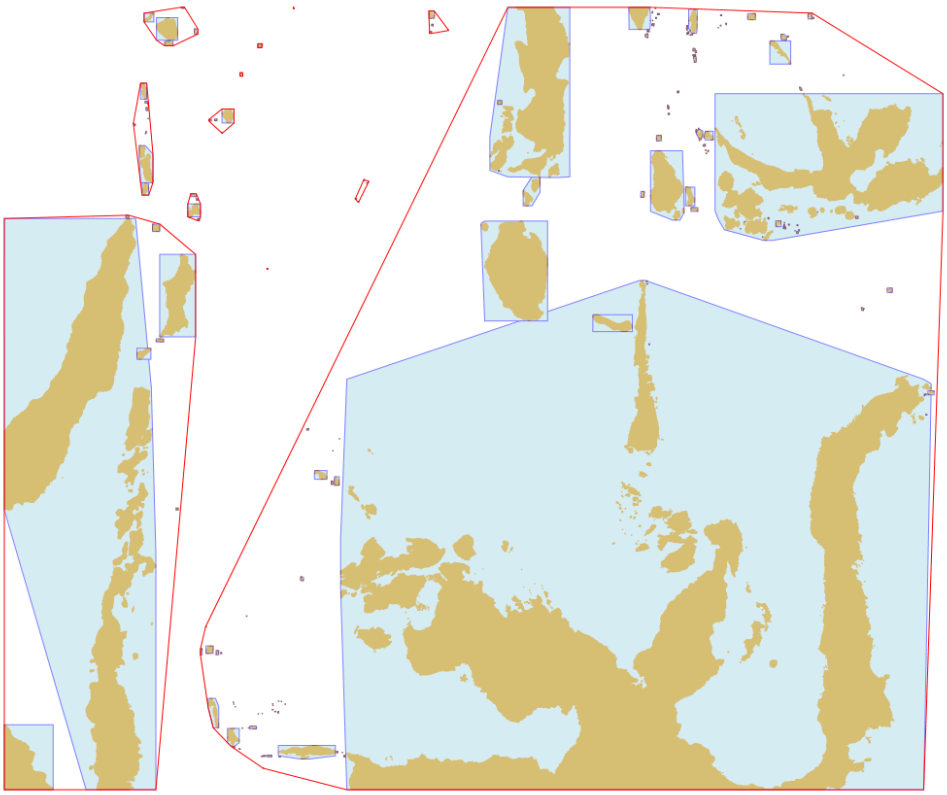

Fig. 3. Families groups.

4 Conclusion

Described algorithm is implemented in C++ and is used as part of route planning implementation. It was tested on real-world data and has demonstrated results presented by figures in this paper. The case depicted had 1031 polygons which were clustered into 310 containers, 201 families and 12 families groups.

It must be noted that a cluster does not imply there is no free movement between its member islands. Figure 3 allows to discern wide passages inside the largest families group. This is a serious drawback from route planning point of view, and it is addressed by another algorithm which further refines the specialized map representation, but is not discussed in this paper.

References

1. V.P. Ilyin, South Ural State University Bulletin **35** 29-40 (2011)
2. T.K. Dobroserdova et al., *IFMBE Proceedings* **6** 403-406 (2015)
3. S.V. Kruglikov, A.S. Kruglikov, Applied Mechanics and Materials. **494-495** 1110-1113 (2014)
4. A. Jain, M. Murty, P. Flynn, ACM Computing Surveys. **31** (1999)
5. P.H. Sneath, R.P. Sokal, *Numerical taxonomy. The principles and practice of numerical classification.* (Freeman, London, UK, 1973).