Finding Optimal Sequences for Area Aggregation—A* vs. Integer Linear Programming

DONGLIANG PENG, Chair of Computer Science I, University of Würzburg, Germany
ALEXANDER WOLFF, Chair of Computer Science I, University of Würzburg, Germany
JAN-HENRIK HAUNERT, Institute of Geodesy and Geoinformation, University of Bonn, Germany

To provide users with maps of different scales and to allow them to zoom in and out without losing context, automatic methods for map generalization are needed. We approach this problem for land-cover maps. Given two land-cover maps at two different scales, we want to find a sequence of small incremental changes that gradually transforms one map into the other. We assume that the two input maps consist of polygons each of which belongs to a given land-cover type. Every polygon on the smaller-scale map is the union of a set of adjacent polygons on the larger-scale map.

In each step of the computed sequence, the smallest area is merged with one of its neighbors. We do not select that neighbor according to a prescribed rule but compute the whole sequence of pairwise merges at once, based on global optimization. We have proved that this problem is NP-hard. We formalize this optimization problem as that of finding a shortest path in a (very large) graph. We present the A* algorithm and integer linear programming to solve this optimization problem. To avoid long computing times, we allow the two methods to return non-optimal results. In addition, we present a greedy algorithm as a benchmark. We tested the three methods with a dataset of the official German topographic database ATKIS. Our main result is that A* finds optimal aggregation sequences for more instances than the other two methods within a given time frame.

CCS Concepts: • **Applied computing** → **Cartography**;

Additional Key Words and Phrases: Continuous map generalization, Land cover, Type change, Compactness, Shortest path

ACM Reference Format:
Dongliang Peng, Alexander Wolff, and Jan-Henrik Haunert. 0000. Finding Optimal Sequences for Area Aggregation—, A* vs. Integer Linear Programming . *ACM Trans. Spatial Algorithms Syst.* 0, 0, Article 00 (0000), 43 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Maps are the main tool to represent geographical information. As geographical information is usually scale-dependent [43, 75], users need to have access to maps at different scales. In order to generate these maps, national mapping agencies produce a base map and then derive maps at smaller scales by *map generalization*. More specifically, map generalization is the process of extracting and arranging geographical information from detailed data in order to produce maps at

Authors' addresses: Dongliang Peng, Chair of Computer Science I, University of Würzburg, Germany, www1.informatik.uni-wuerzburg.de/peng; Alexander Wolff, Chair of Computer Science I, University of Würzburg, Germany, www1.informatik.uni-wuerzburg.de/wolff; Jan-Henrik Haunert, Institute of Geodesy and Geoinformation, University of Bonn, Germany, www.geoinfo.uni-bonn.de/haunert.

2374-0353/0000/0-ART00 $15.00
https://doi.org/0000001.0000001

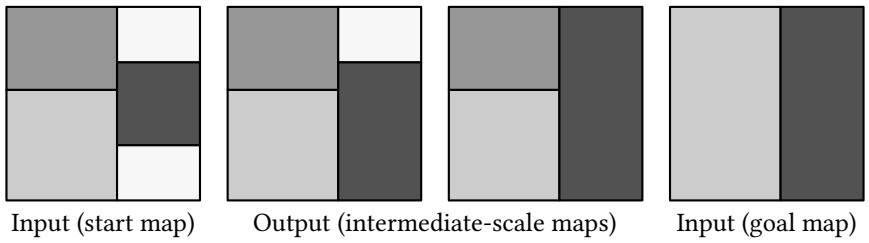


Fig. 1. The input and a possible output for an instance of our problem.

smaller scales. A requirement of map generalization is to emphasize the essential while suppress the unimportant, and at the same time maintain logical relationship between objects [75]. As manual generalization is labor-intensive [15], automating map generalization is a promising way to produce up-to-date maps at high speed and low cost [39].

In our digital age, people interactively read maps on computers and mobile phones. An often used interaction is zooming. When users zoom in or out, a map must be changed to provide information appropriate to the corresponding zoom level. However, large discrete changes may distract users. The *on-the-fly generalization* [76], which generalizes map features (e.g., polyglines and polygons) in real time, can mitigate this problem. Still, large discrete changes can be introduced. By a usability test, Midtbø and Nordvik [41] have shown that a map is easier to follow if the map extent transits smoothly than stepwise. In addition to smoothly transiting map extent, we want to change also map features smoothly when users are zooming. We believe that this strategy will allow users to follow maps even more easily. The process of producing maps with a smooth scale transition is known as *continuous map generalization* (CMG), or simply *continuous generalization*. Ideally, there should be no discrete change in CMG. However, the term is also used when the discrete changes are small enough not to be noticed as, e.g., Šuba et al. [63] stated.

A simple way of realizing CMG is to fade in and fade out raster maps as did by Pantazis et al. [46], but it is not at all clear whether such animations would improve the experience of map users. Furthermore, although it is possible to generalize raster maps, we consider it more straightforward to work on vector maps. For example, Jaakkola [31] generalized raster maps, but he used essentially the same object-based operators that are common in the map generalization of vector data. He even converted raster data to vector data when he wanted to simplify the boundaries of the land-cover areas. Therefore, our paper will work on a vector map of land-cover areas.

Problem definition. In this paper, we deal with CMG of land-cover areas. The land-cover area is of significant importance on maps. When users zoom out, some land-cover areas become too tiny to be seen, which result in visual clutter. In order to provide users with good visual experience during zooming operations, we propose to aggregate the tiny areas into their neighboring areas.

A *land-cover map* is a planar subdivision in which each area belongs to a land-cover class or *type*. Suppose that there are two land-cover maps of different scales that cover the same spatial region. We consider the problem of finding a sequence of small incremental changes that gradually transforms the larger-scale map (the *start map*) to the smaller-scale map (the *goal map*). We use this sequence to generate and show land-cover maps at intermediate scales (see Figure 1). In this way, we try to avoid large and discrete changes during zooming.

With the same motivation, a strategy of hierarchical schemes has been proposed. This strategy generalizes a more-detailed representation to obtain a less-detailed representation based on small incremental changes, e.g., the Generalized Area Partitioning tree (GAP-tree). This tree can be constructed if only the larger-scale map is given [70] or if both the larger-scale map and the smaller-scale map are given [23]. Typically, the next change in such a sequence is determined locally, in a greedy fashion. If we insist on finding a sequence that is optimal according to some global measure, the problem becomes complicated.

We assume that there exist many-to-one relationships between the areas of the start map and the areas of the goal map. This assumption is based on the fact that many algorithms [e.g., 27, 45, 74] result in many-to-one relationships when aggregating land-cover areas. Their inputs and generalized results together can be used as our inputs. However, we should not use those algorithms to generate a sequence of maps at different scales because those algorithms do not take into account the consistence between the generated maps. We use both a start map and a goal map instead of using only the start map because our generated maps at intermediate scales should be able to benefit from a goal map with high quality.

We term the areas of the goal map *regions*. That is, every region is the union of a set of areas on the start map. The type of a region may differ from the types of its components. For example, a small water area together with multiple adjacent forest areas may constitute a large forest region on the goal map. However, we assume that every region, on the goal map, contains at least one area of the same type on the start map. Our assumptions hold if the goal map has been produced with an automatic method for area aggregation, for example, by the method of Haunert and Wolff [27]. That method produces a land-cover map at a single smaller scale, given a land-cover map at a larger scale. Although Haunert and Wolff [27] attain results of high quality, they do not produce a sequence of maps.

Our method can also be extended to find an aggregation sequence for two maps (a start map and a goal map) that are from different sources. In that case, one could compute a map overlay of the two maps and use the result (with combined boundaries from both input maps and land cover classes from the given larger-scale map) as the start map.

In this paper, we try to find an optimal sequence to aggregate the land-cover areas on the start map one by one until we arrive at the goal map. We first independently deal with each region of the goal map (with its components on the start map). Once we have found an aggregation sequence for each region, we integrate all the sequences into an overall sequence, which transforms the start map into the goal map (see Figure 2). Our aggregation sequence may be cooperated with the GAP-face tree [70], the map cube model [65], or ScaleMaster [2, 68], to support on-the-fly visualization. Smoothly (dis-)appearing areas can be realized by integrating our results into the *space-scale cube* [71, 72].

Contribution. We first show some related work (Section 2). Then, we formally model our problem, analyze the size of our model in a worst-case scenario, introduce our methods, and present the basic concepts of our method (Section 3). We define our cost functions (Section 4). We prove that our problem is NP-hard (Section 5). Then, we develop and compare three methods for finding aggregation sequences. First, we present a greedy algorithm (Section 6). Second, we develop a new global optimization approach based on the A* algorithm (Section 7). Third, we model our pathfinding problem as an *integer linear program* (ILP), and we solve this ILP with minimizing our cost function (Section 8). Our ILP uses binary (0–1) variables. These variables help us to model our problem, but in general, it is NP-hard to solve an ILP optimally. By comparing with the greedy

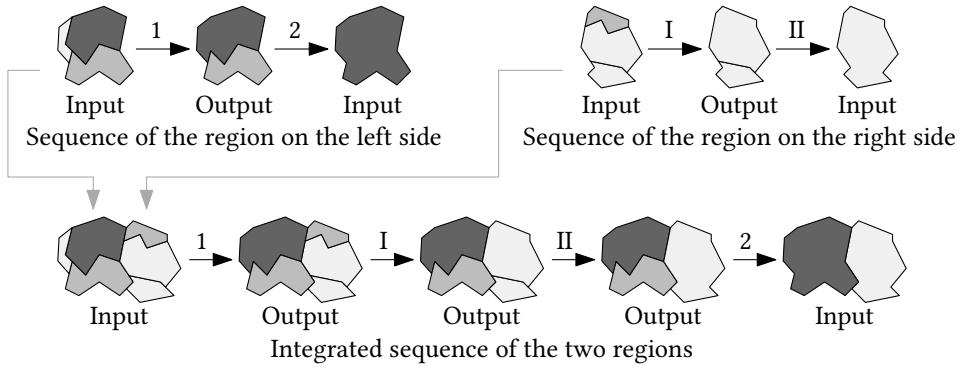


Fig. 2. Integrating two aggregation sequences of different regions: the resulting sequence contains the given sequences as subsequences and always takes the subdivision with smallest patch next. The gray arrows show the integration of the two regions.

algorithm, which is used as a benchmark, we are able to see whether A* or the ILP-based algorithm, which are more complex and slower, indeed perform better. Our case study uses a dataset of the German topographic database ATKIS (Section 9). In the concluding remarks, we show possible ways to improve our methods (Section 10).

We do not deal with simplifying polylines in this paper. The simplification can be handled separately from the aggregation of areas, for example, by using one method of Douglas and Peucker [14], Saalfeld [56], or Wu et al. [81]. Those methods can be used to set up the binary line generalisation tree (BLG-tree) of van Oosterom and Schenkelaars [73], which is a hierarchical data structure that defines a gradual line simplification process. Note that simplifying polylines will not speed up our three algorithms. The reason is that we compute the areas of the polygons and the lengths of the boundaries in a preprocessing. These values will be associated with a graph representation of the dataset, where polygons are represented as vertices and a boundary separating two polygons (no matter how detailed it is) is represented with a single edge. During computation, the three algorithms do not need to go through the vertices of the land-cover areas.

Although splitting polygons is a good step of generalizing land-cover areas, we do not integrate it into our method at this moment. Some examples of splitting are as follows. Smith et al. [61] and Thiemann and Sester [64] proposed to split tiny polygons and then to merge the split parts into the neighboring polygons. Meijers et al. [40] developed an algorithm to split a polygon (the splittee) based on a constrained Delaunay triangulation. During splitting, their algorithm is capable of taking into account the attractivenesses between the splittee and its neighbors; when merging, a more attractive neighbor will get a larger portion from the splittee.

2 RELATED WORK

2.1 Continuous map generalization

Continuous map generalization (CMG) has received a lot of attention from cartographers and computer scientists. Van Kreveld [69] proposed five gradual changes to support the continuous zooming of maps, which are *moving*, *rotating*, *morphing*, *fading*, and *appearing*. He suggested using these gradual changes to adapt discrete generalization operators for CMG. Sester and Brenner [59] suggested simplifying building footprints based on small incremental steps and to animate each step smoothly. Li and Zhou [37] built hierarchies of road segments, which they then used to omit road segments from lower levels of the hierarchy. Moreover, they evaluated similarities

between their results and existing maps. Peng and Touya [50] gradually grew buildings to built-up areas by aggregating buildings whenever they become too close. Touya and Dumont [67] progressively replaced buildings with blocks. In addition, their method automatically inferred landmarks and put the landmarks on top of the blocks. Šuba et al. [63] continuously generalized road networks which are represented as a set of areas. Their method repeatedly finds the least-important area and then either merges it with an adjacent area or collapses it to a line segment. Danciger et al. [10] investigated the growing of regions, while preserving topology, area ratios, and relative positions. The strategy of using two maps at different scales to generate intermediate-scale maps has been studied in multiple representations, e.g., with respect to the selection of roads or rivers [18, 48]. Actually, this strategy is the key idea of the morphing-based methods for CMG. In order to morph from one polyline to another polyline, which respectively represent, say, roads on a larger-scale map and a smaller-scale map, we first need to compute corresponding points between them [e.g., 4, 5, 11, 34, 35, 44]. Then morphing can be realized by interpolating a set of intermediate polylines. Nöllenburg et al. [44] computed an optimum correspondence between two given polylines according to some cost function. While straight-line trajectories are often used for interpolation [e.g., 4, 11], Whited and Rossignac [77] considered four other alternatives, i.e., *hat*, *tangent*, *circular*, and *parabolic* paths based on so-called *ball-map* [5]. Peng et al. [49] defined trajectories requiring that angles in vertices and edge lengths should change linearly. As these requirements may not agree with each other, their method mediates between them using least-squares adjustments. Peng et al. [51] morphed county boundaries to provincial boundaries. For county boundaries that do not have corresponding provincial boundaries, they generated the correspondences based on *compatible triangulations*. Van Oosterom and Meijers [71] used a data structure called *smooth topological generalized area partitioning* to support visualizing CMG. One of their contributions is that a polygon is merged into another polygon continuously by moving the boundary of the former. Huang et al. [30] proposed a matrix-based structure to support CMG, using a river network as an example. For a given scale, their structure yields the rivers that should be kept as well as how much these rivers should be simplified.

2.2 Optimization in map generalization

Map generalization generally specifies and takes into account requirements in order to produce maps of high quality [62]. We categorize requirements as hard and soft constraints. For example, when users zoom out, some land-cover areas become too small to be seen. These areas need to be aggregated. When we aggregate one area into another, the type of the former is changed to the type of the latter. In this problem, a hard constraint could be that we aggregate only two polygons at each step in order to keep changes small (see for example Figure 1). A soft constraint could be that we wish to minimize the changes of types, e.g., we prefer aggregating a grass area into a farm area rather than into a settlement area. This is a typical *optimization* problem, where we stick to hard constraints and try to fulfill soft ones as well as possible. Optimization for map generalization is important not only because it finds optimal solutions, but also because it helps us to evaluate the quality of a model [25, 28, 29]. When we wish to minimize the changes of types in aggregating areas one by one, a model could be to minimize the *greatest* change over all the steps. Using a greedy algorithm, we can minimize the change at each step, but the result does not necessarily minimize the greatest change over all the steps. If a result is bad, we cannot tell if the bad result comes from the model or from the greedy algorithm. Using optimization, we are able to find optimal solutions of the model at least for small instances. If even an optimal solution is bad, then we can exclude that the bad result is from the greedy algorithm. That is to say, the bad result is because of the model. In this case, we should improve the model; we may want to minimize the *average* change over all the steps. Moreover, optimization is useful for evaluating heuristics. We need heuristics because many

optimization problems cannot be solved efficiently [e.g., 24, 27]. While heuristics can find some solutions in reasonable time, it is important to know the quality of these solutions. Fortunately, we can often find an optimal solution when the size of an instance is sufficiently small. Consequently, we are able to evaluate the quality of a heuristic by comparing its results with optimal solutions on small instances.

Optimization has been widely used in map generalization. For example, Harrie [19] displaced objects based on least-squares adjustments (LSA) to solve spatial conflicts. In his problem, the soft constraints for shapes and locations may contradict each other. Therefore, it is necessary to mediate between these constraints, which can be done by LSA. Sester [58] used LSA not only for displacing objects but also for simplifying buildings. She required that the output boundaries should be as close to the original buildings as possible. Tong et al. [66] generalized land-cover areas, where LSA was used to preserve the sizes of the land-cover areas. Regnauld [55] grouped buildings based on minimum spanning trees in order to typify the buildings in a group. Burghardt [3] smoothed lines based on energy minimization. According to his setting, a line contains less energy if it is smooth and close to the original line. He repeatedly displaced the line until a stable state in terms of minimizing his energy function is found. Haunert and Wolff [27] aggregated land-cover areas based on mixed-integer linear programming to generate a map at a target scale. Their method is based on a global optimization. They minimize a combination of type changes and cost for non-compact shapes while satisfying constraints on the sizes of the output regions. Haunert and Wolff [26] simplified building footprints by solving an integer linear program (ILP). They aimed at minimizing the number of edges in the output under the restriction that the simplified buildings must be topologically safe, that is, the selected and extended edges must not intersect with each other. Oehrlein and Haunert [45] aggregated the departments of France according to unemployment rates based on integer linear programming; they used a cutting-plane method to speed up solving their ILP. Funke et al. [17] simplified administrative boundaries based on an ILP. Their aim was to minimize the number of edges while keeping the result boundaries close to the original ones and avoiding any intersection. At the same time, they required that every city, represented by a point, stays in the same face as before the generalization.

2.3 Optimization in *continuous* map generalization

Optimization becomes more delicate when we deal with CMG. In this field, we have requirements not only for a specific map but also for relations between maps at difference scales. Some optimization techniques have been applied to CMG. In the aforementioned article, Nöllenburg et al. [44] used dynamic programming to match points of two polylines to support morphing according to some matching cost. Schwartges et al. [57] used mixed-integer linear programming to select points of interest. They required that a point, once disappeared, should not show up again during zooming out. They also required that any two points should be sufficiently far away from each other. Based on these requirements, they wanted to show as many points as possible for a given scale interval. Chimani et al. [7] computed a deletion sequence for a road network by integer linear programming and efficient approximation algorithms. They wanted to delete a stroke, which is a sequence of edges, at each step while keeping the remaining network connected. They assigned each edge a weight, and their objective was to maximize the total weight over all the road networks of all the steps. Peng et al. [49] defined trajectories based on LSA for morphing between polylines, where LSA is used to mediate between the requirements for angles and edge lengths of the intermediate polylines.

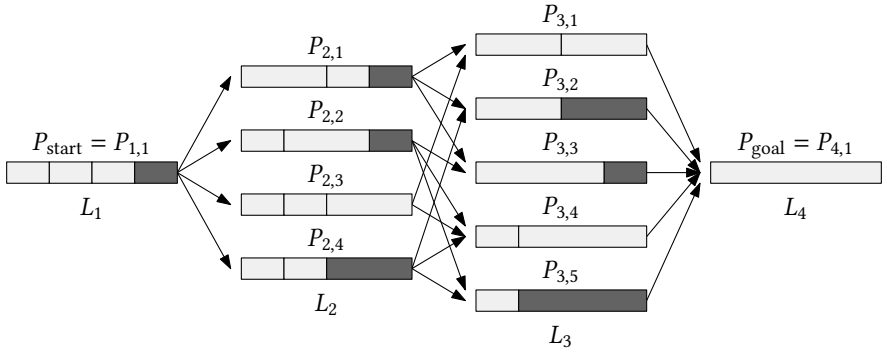


Fig. 3. The subdivision graph, G_S. The nodes of the graph are the subdivisions. There is an arc from subdivision $P_{t,i}$ to subdivision $P_{t+1,j}$ if $P_{t+1,j}$ is the result of an aggregation step from $P_{t,i}$.

3 PRELIMINARIES

We show how to compute an aggregation sequence for a single region, R. For a goal map with many regions, we "interleave" the sequences for each of them with respect to the order of the smallest patches (see for example Figure 2). This integration is similar to the merge step in the Mergesort algorithm; see Cormen et al. [8, Section 2.3]. To allow us to describe our method more easily, below we assume that the goal map has only one region. This region consists of n land-cover areas (components) on the start map. In other words, the union of the n land-cover areas is the only region on the goal map.

To find a sequence of small changes that transforms the start map into the goal map, we require that every change involves only two areas of the current map. More precisely, in each step the smallest area u is merged with one of its neighbors v (v does not have to be the smallest neighbor) such that u and v are replaced by their union. The type of the union is restricted to be the type of either u or v. If the union uses the type of u, we say that area v is *aggregated into* area u, and vice versa. How to aggregate exactly is decided by optimizing a global cost function (see Section 4). This requirement ensures that the size of the smallest area on the map increases in each step. Hence, the sequence reflects a gradual reduction of the map's scale. From another perspective, we consider the smallest area as the least important, rather than involving more rules for (non-)importance. Even though the requirement reduces the number of possible solutions, there is still enough room for optimization since we leave open with which of its neighbors the smallest area is aggregated. We term a sequence of changes that adheres to our smallest-first requirement simply an *aggregation sequence*.

3.1 Model

We consider a directed graph G_S, which we call the *subdivision graph* (see Figure 3). The node set V_S of G_S contains nodes for all the possible maps (or *subdivisions*), including the start map, all possible intermediate-scale maps, and the goal map. The arc set E_S of G_S contains an arc $(P_{t,i}, P_{t+1,j})$ between any two maps $P_{t,i}$ and $P_{t+1,j}$ in V_S if $P_{t+1,j}$ can be reached from $P_{t,i}$ with a single aggregation operation, involving a smallest area. On this basis, any directed path in G_S from the start map to the goal map defines a possible aggregation sequence.

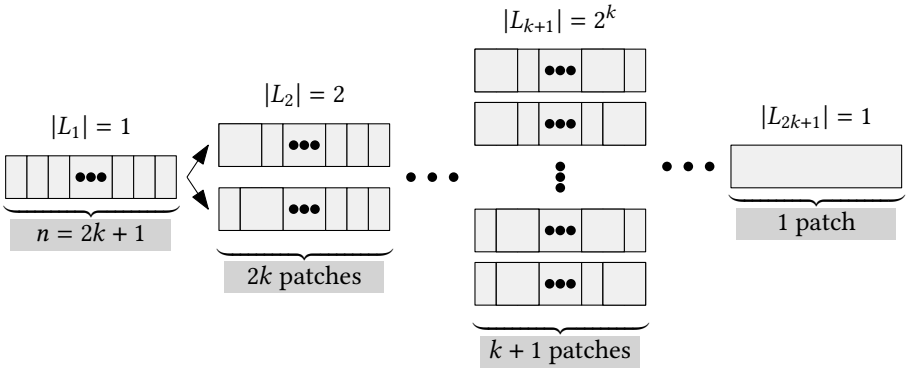


Fig. 4. An example to show that the size of subdivision graph G_S has exponential lower bound.

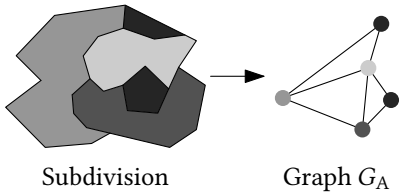


Fig. 5. The graph of a subdivision. Each polygon of the subdivision is represented as a node in the graph. There is an edge between two nodes if the corresponding two polygons are adjacent.

3.2 Notation

We represent each land-cover area by a polygon with a type. We denote by P the set of polygons on the start map. We use p, q, r, or o to denote polygons. A *patch* is a set of polygons whose union is connected. Each patch also has a unique land-cover type. We use u or v to denote the patches.

Recall that we are dealing with a single region and there are n land-cover areas on the start map in this region. Hence, the desired aggregation sequence consists of $n - 1$ steps. There are n subdivisions on a path from the start map to the goal map. We use $t \in T = \{1, 2, \ldots, n\}$ to denote *time*. When $t = 1$, the subdivision consists of n patches, and there is only one patch remaining when $t = n$. The subdivision graph consists of layers $L_1, \ldots, L_n$, where layer $L_t = \{P_{t,1}, \ldots, P_{t,n_t}\}$ contains every possible subdivision $P_{t,i}$ with $n - t + 1$ patches (see Figure 4).

Sometimes, we need a graph to represent the adjacencies of the land-cover areas in a subdivision, we call such a graph G_A (see Figure 5).

3.3 Exponential lower bound

We now analyze the size of subdivision graph G_S. Our analysis is inspired by Keane [33], where we use a row of n land-cover areas. In our instance (see Figure 4), the start map consists of $n = 2k + 1$ rectangular patches, and the goal map is simply the union of the n patches. The patches have area sizes $100 + \frac{1}{n}, 99 + \frac{2}{n}, 100 + \frac{3}{n}, 99 + \frac{4}{n}, \ldots, 99 + \frac{n-1}{n}$, and 101, from left to right. According to our rule, we always aggregate the smallest patch with one of its neighbors. Therefore, in the first k steps from the start map, we aggregate every other patch with one of its neighbors. However, we do

not know which one is the right choice at each of the steps in order to minimize our costs (see Section 4). We have to try both of the two choices, aggregating with the left patch or with the right one. As a result, there are $2^k = 2^{(n-1)/2}$ subdivisions in layer L_{k+1}. That is to say, the size of subdivision graph G_S has exponential lower bound.

3.4 Methods

Our idea is to obtain an optimal aggregation sequence through computing a path with minimum weight, from the start to the goal (see Figure 3). This idea obviously requires that the arc weights are set such that a minimum-weight start–goal path does actually correspond to an aggregation sequence of maximum cartographic quality. Moreover, putting the idea to practice is far from trivial since graph G_S can be huge. We compare a greedy algorithm, A*, and an ILP-based algorithm in finding such paths. Note that our inputs are only subdivisions P_{start} and P_{goal} (see Figure 3). We generate a subdivision (node) only when we want to visit it.

In directed acyclic graphs, shortest paths can be found slightly faster than in general directed or undirected graphs. An off-the-shelf shortest-path algorithm for directed acyclic graphs (e.g., Cormen et al. [8, Section 25.2]), however, will explore the whole graph, which has exponential size. The A* algorithm can be seen as a refinement of Dijkstra's algorithm, which for a user-specified given source computes shortest paths to all other nodes in an edge-weighted graph [12]. Even when using Dijkstra's algorithm to compute only a single shortest path to a user-specified destination, a large number of nodes need to be explored. The same holds for shortest-path algorithms that make use of a topological order of the nodes in a directed acyclic graph. Compared to these algorithms, the A* algorithm can greatly reduce the number of explored nodes. The challenge in our work was to tune the A* algorithm such that it explores only a small fraction of the graph.

4 COST FUNCTIONS

Figure 3 shows that there are many ways to aggregate from the start map to the goal map. Apparently, some of the ways are more reasonable than others. In our example, we consider sequence $(P_{1,1}, P_{2,1}, P_{3,1}, P_{4,1})$ more reasonable than sequence $(P_{1,1}, P_{2,4}, P_{3,5}, P_{4,1})$. This is because that the dark area should not expand so much when the target color is light gray. We want to provide users with a most reasonable sequence because we believe that an unreasonable sequence irritates users. To find a most reasonable sequence, we introduce cost functions. In the cost functions, we charge a higher penalty when an aggregation step is less reasonable. As a result, by minimizing the overall cost of an aggregation sequence, we find a most reasonable sequence.

It is difficult to define what *reasonable* exactly means because users may have different preferences. Four preferences have been discussed by Cheng and Li [6]; see Figure 6. A small land-cover area can be aggregated into another area that isolates the area (Figure 6b), that is the largest neighbor (Figure 6c), that shares the longest boundary (Figure 6d), or that has the most similar type (Figure 6e). To keep our aggregation problem independent of user preferences, our cost function takes two aspects into account: one based on semantics and the other based on shape. In terms of semantics, we require that the *type* of a land-cover area changes as little as possible. This requirement means that we prefer, for example, aggregating an area with type *swamp* into an area with type *wet ground* rather than into an area with type *city*. In terms of shape, we hope to have areas which are as *compact* as possible. Our argument is that an area is easier to be identified by a human being if it is more compact (less clutter). We also consider the total *length* of the interior boundaries as an alternative compactness; we consider subdivision $P_{t,i}$ as more compact than subdivision $P_{t,i'}$ if the total length of the interior boundaries of $P_{t,i}$ is less than that of $P_{t,i'}$. We add this alternative because we want to make a comparison involving an ILP, where a *linear* cost function must be used. Note

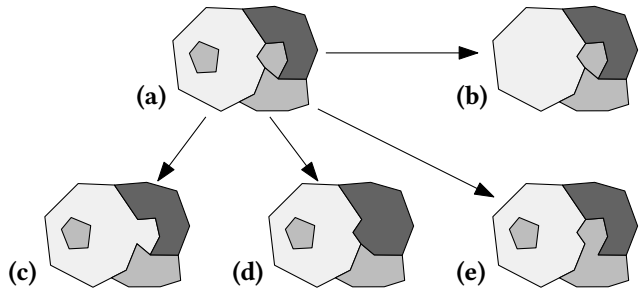


Fig. 6. Aggregating land-cover areas according to different preferences by Cheng and Li [6]. Aggregating a small land-cover area into another one that isolates the area (b), that is the largest neighbor (c), that shares the longest boundary (d), or that has the most similar type (e).

that most compactness measures are *not* linear; for example, see Li et al. [36] and Maceachren [38]. Although the length of interior boundaries is not sufficiently precise to describe compactness [82], it is often used as a fair alternative when compactness is considered in an ILP [e.g., 42, 80]. Haunert and Wolff [27] employed the centroids of a set of land-cover areas. One of their costs is the sum of the distances from the centroids to a *reference point*. The reference point is one of the centroids that minimizes the sum. The sum of the distances can be computed linearly. We use the length of interior boundaries instead of the distance of centroids because the former is more relevant to the shapes of the patches. Also, Harrie et al. [20] showed that longer lines generally yield lower map readability.

4.1 Cost of type change

Suppose that we are at the step of aggregating from subdivision $P_{s,i}$ to subdivision $P_{s+1,j}$. In this step, patch u is aggregated into patch v (see Figures 7a and 7b). We denote the types of the two patches by $T(u)$ and $T(v)$. We define the cost of type change of this step by

$$f_{\text{type}}(P_{s,i}, P_{s+1,j}) = \frac{A_u}{A_R} \cdot \frac{d_{\text{type}}(T(u), T(v))}{d_{\text{type\_max}}}, \tag{1}$$

where A_u is the area of patch u, and A_R is the area of region R (see Section 3). We use A_R and $d_{\text{type\_max}}$ to normalize the cost of type change. Constant $d_{\text{type\_max}}$, the maximum cost over all type changes, is known from the input. The input specifies cost $d_{\text{type}}(T_1, T_2)$ of changing type T_1 to type T_2. Specifically, we denote by T_{goal} the type of the patch on the goal map. For simplicity, we use a metric as the cost function of type change (see Section 9). A metric distance is symmetric, which is different from the asymmetric one used in Dilo et al. [13]. In their definition, for example, the distance from type *building* to type *road* is 0.5, but the distance from *road* to *building* is 0.

For path $\Pi = (P_{1,i_1}, P_{2,i_2}, \ldots, P_{t,i_t})$, we define the cost of type change over the steps by

$$g_{\text{type}}(\Pi) = \sum_{s=1}^{t-1} f_{\text{type}}(P_{s,i_s}, P_{s+1,i_{s+1}}). \tag{2}$$

4.2 Cost of compactness

We use the compactness definition of Frolov [16], i.e., the compactness value of a patch, say, u is

$$c(u) = \frac{2\sqrt{\pi A_u}}{l_u}, \tag{3}$$

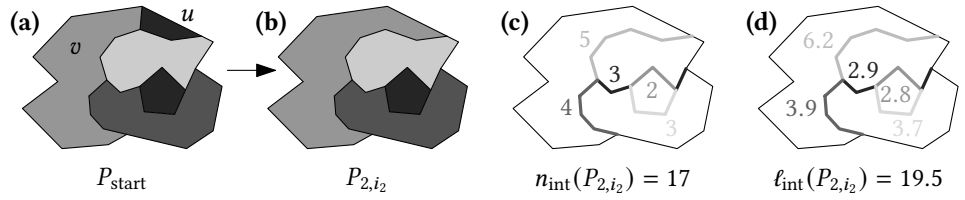


Fig. 7. An aggregation step, where patch u, the dark patch at the top, is aggregated into patch v. Figures (c) and (d) respectively show the number of edges and the lengths of the interior polylines after the aggregation.

where A_u and l_u are the area and the perimeter. For a subdivision $P_{s,i}$, we denote by $C(P_{s,i})$ the set of the patches' compactness values.

We wish to maximize the sum of the average compactness values over all intermediate maps, while our objective will be minimizing a cost function. To adapt the average compactness to our methods, we define and minimize a cost related to compactness. Recalling that there are $n - s + 1$ patches at time s, we define the cost of compactness for subdivision $P_{s,i}$ as

$$f_{\text{comp}}(P_{s,i}) = \frac{1 - \frac{1}{n-s+1} \sum_{c \in C(P_{s,i})} c}{n-2}, \tag{4}$$

where we use values $n - s + 1$ and $n - 2$ to normalize the cost of compactness.

For path Π (see Section 4.1), we define the cost of compactness over all intermediate maps (that is, neglecting $P_{1,1}$ and the last subdivision in the path) by

$$g_{\text{comp}}(\Pi) = \sum_{s=2}^{t-1} f_{\text{comp}}(P_{s,i_s}). \tag{5}$$

4.3 Cost of length

We denote the set of interior boundaries for a subdivision $P_{s,i}$ by $B(P_{s,i})$. The cost in terms of interior length of this subdivision is defined as

$$f_{\text{lgth}}(P_{s,i}) = \frac{\left(\sum_{b \in B(P_{s,i})} |b|\right) / D(s)}{n-2}, \tag{6}$$

where

$$D(s) = \frac{n-s}{n-1} \sum_{b \in B(P_{\text{start}})} |b|. \tag{7}$$

Function $D(s)$ computes the "expected" total length of the interior boundaries at time s, where we expect that this total length decreases linearly according to time s. In special, $D(1) = \sum_{b \in B(P_{\text{start}})} |b|$ and $D(n) = 0$. Similarly to Equation 4, we use $D(s)$ and $n - 2$ to normalize the cost of length.

For path Π (see Section 4.1), we define the cost of length over all intermediate maps (that is, neglecting $P_{1,1}$ and the last subdivision in the path) by

$$g_{\text{lgth}}(\Pi) = \sum_{s=2}^{t-1} f_{\text{lgth}}(P_{s,i_s}). \tag{8}$$

Note that in theory, a patch, u, with a small perimeter can be extremely non-compact according to measure $c(u)$ in Equation 3, thus the two measures, f_{comp} and f_{lgth}, are not interchangeable. However, if we assume that all areas of the map have the same size (i.e., A_u of Equation 3 is a constant), it would make no difference whether we minimize an area's perimeter or maximize the

area's compactness $c(u)$. Obviously, the areas in our dataset have different sizes. However, since we iteratively remove the smallest area, the differences do not become too large. Therefore, measuring the overall compactness of a map based on the total length of all the interior boundaries is quite reasonable.

4.4 Combining cost functions

When we generate a sequence of intermediate-scale maps, we want to change the types of the land-cover areas as little as possible and want to have compact land-cover areas. Therefore, we combine the cost of type change (Equation 2) and the cost of compactness (Equation 5). That is,

$$g_1(\Pi) = (1-\lambda)g_{\text{type}}(\Pi) + \lambda g_{\text{comp}}(\Pi), \tag{9}$$

where $\lambda \in [0,1]$ is a parameter to assign importances of f_{type} and f_{comp}. We simply use $\lambda = 0.5$ in our experiments. We want to find a path Π from P_{start} to $P_{t,i}$ that minimizes, among all such paths, $g_1(\Pi)$. Slightly abusing notation, we denote the cost of an optimal path from P_{start} to $P_{t,i}$ by $g_1(P_{t,i})$. Using $g_1(P_{t,i})$, we compare a greedy algorithm and A* in finding optimal sequences for area aggregation.

As said before, we want to make a comparison involving integer linear programming while our cost of compactness (see Equation 3) cannot be computed linearly in an integer linear program. Therefore, we combine the cost of type change (Equation 2) and the cost of length (Equation 8), which can be computed linearly in an integer linear program. That is,

$$g_2(\Pi) = (1-\lambda)g_{\text{type}}(\Pi) + \lambda g_{\text{lgth}}(\Pi). \tag{10}$$

We compare the greedy algorithm, A*, and an ILP-based algorithm using g_2.

5 NP-HARDNESS PROOF

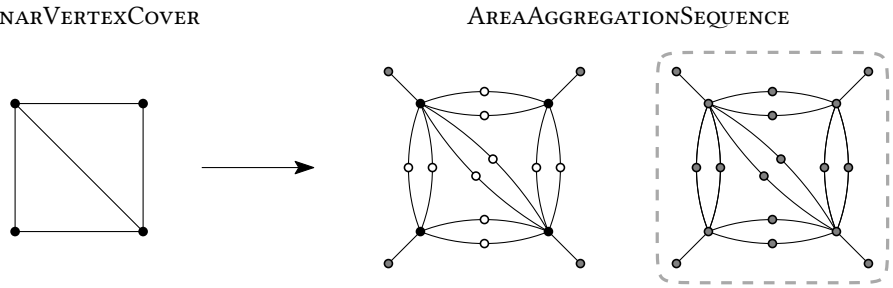
Although we have shown that the graph of subdivisions has an exponential size (see Section 3), one may develop a clever algorithm to find an optimal sequence efficiently. In the following, we prove that finding such a sequence is indeed NP-hard. In the proof, we neglect the cost of compactness or the cost of length. Considering one of the two costs will make the computation even more difficult.

Theorem 1. *AreaAggregationSequence is NP-hard even if we only consider the cost of type change.*

Proof. Our NP-hardness proof is by reduction from the NP-complete problem PlanarVertexCover, which is to decide, for a given planar graph $G_{\text{A}} = (V_{\text{A}}, E_{\text{A}})$, whether there exists a vertex cover with at most a given number k_{A} of vertices. For an instance of PlanarVertexCover (Figure 8a), we define a corresponding instance of AreaAggregationSequence whose start map consists of gray, black, and white areas and whose goal map consists of only one large gray patch. The adjacency graphs of the two maps are illustrated in Figures 8b and 8c, where the colors of the vertices represent the colors of the corresponding areas. More precisely, for each vertex of G_{A} in Figure 8a, we define a gray vertex and a black vertex, which we connect with an edge. For each edge $\{u, v\}$ of G_{A}, we define two white vertices and connect each of them both with the black vertex for u and the black vertex for v (Figure 8b).

We define the weights of the vertices in Figure 8b as follows:

- Every white vertex has weight 1.
- Every black vertex has weight 2.
- Every gray vertex has weight $2|V_{\text{A}}| + 2|E_{\text{A}}|$, which is equal to the total weight of all white and black vertices.



(a) $G_A = (V_A, E_A)$ **(b)** Start map **(c)** Goal map

Fig. 8. Our NP-hardness reduction. The dashed area in (c) represents the vertices that are merged.

When we merge two vertices, the weight of the new vertex is the sum of the two. In each aggregation step, we require the smallest area to be aggregated with one of its neighbors. The area sizes correspond to the weights of the vertices in Figure 8b. Therefore, our definition of the weights to a certain degree determines the order in which the vertices are selected:

Phase I: In each of the first $2|E_A|$ steps, a white vertex is selected and merged with one of its neighbors, such that the white vertex receives the neighbor's color or vice versa (Figure 9 shows a possible result).

Phase II: In each of the next $|V_A|$ steps, a non-gray vertex is selected and merged with one of its neighbors.

Phase III: $|V_A| - 1$ steps remain to reach the goal map.

To complete the reduction, we need to define the costs of color changes. For any color change, we charge one unit of cost per unit weight. Due to our construction, Phases II and III can be accomplished with a total cost of $2|V_A| + 2|E_A|$, no matter how Phase I is accomplished. This is because, after Phase I, every non-gray patch will be adjacent to a gray patch (vertex). Thus, if any non-gray patch becomes selected in Phase II, it can be aggregated with a gray patch (vertex) and receive the color gray. This implies that Phase II costs $2|V_A| + 2|E_A|$ (which is equal to the total weight of all initially white and black vertices) and, since after Phase II all patches are gray, Phase III does not cause any additional cost. It is also clear that there is no cheaper way to accomplish Phases II and III, since it is impossible to color a vertex gray in Phase I. Consequently, since the total cost of Phases II and III is fixed, it is only interesting to ask at which cost Phase I can be accomplished. It turns out that, if G_A has a vertex cover $C_A \subseteq V_A$, then Phase I can be accomplished with cost $2|C_A|$; only the black vertices corresponding to vertices in C_A need to change their color from black to white, and each of them has weight 2 (see Figure 9). To summarize, if graph G_A has a vertex cover C_A, then the corresponding instance of AreaAggregationSequence has a solution of total cost $2|C_A| + 2|V_A| + 2|E_A|$.

It remains to be shown that, if C_A^* is a minimum vertex cover of G_A, then there is no solution with total cost less than $2|C_A^*| + 2|V_A| + 2|E_A|$. To see why, we assume that such a solution exists. If we keep the black color of a vertex from C_A^* (the total cost decreases by 2), then we will need to at least change two white vertices to black vertices (the total cost increases by 2 at least). Therefore, we have found a contradiction to our assumption.

□

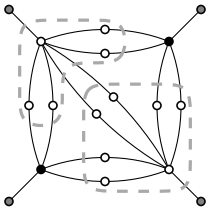


Fig. 9. The situation after Phase I has been conducted such that all black vertices corresponding to a vertex cover of G_A have been recolored white. The dashed areas represent the vertices that are merged.

6 A GREEDY ALGORITHM

A motivation for the greedy algorithm is that a very similar iterative algorithm has been used by van Oosterom [70] for constructing the tGAP data structure. However, we have to make minor modifications to ensure that the computed aggregation sequence ends with the goal map that, in our situation, is given as a part of the input. We use our greedy algorithm as a benchmark so that we are able to see whether the A* algorithm or the ILP-based algorithm indeed perform better.

At any time t, our greedy algorithm aggregates the smallest patch with one of its neighbors. We pick the neighbor in a greedy way. We suppose that the smallest patch, u, has k_u neighbors, then there are $2k_u$ ways to aggregate (when we aggregate a patch into another patch, the union uses the type of the latter). In order to guarantee that our final result (e.g., the polygon of layer L_4 in Figure 3) will have the type of T_{goal}, we add one more rule to our greedy algorithm. Suppose that patch v is one of u's neighbors. The greedy algorithm aggregates u into v if the type distances fulfill that $d_{\text{type}}\left(T(u), T_{\text{goal}}\right) \geq d_{\text{type}}\left(T(v), T_{\text{goal}}\right)$; otherwise, the algorithm aggregates v into u. This rule excludes, say, k_e aggregation choices, and we have $2k_u - k_e$ choices left. Then we compute the costs for each of the $2k_u - k_e$ aggregation choices and select the aggregation that has the least cost. In other words, we aggregate the smallest patch with its most *compatible* neighbor.

In accordance with our two combinatorial costs in Section 4.4, we define two cost functions. Suppose that we are at the step of aggregating from subdivision $P_{s,i}$ to subdivision $P_{s+1,j}$. The first cost function is

$$f_1(P_{s,i}, P_{s+1,j}) = (1-\lambda) f_{\text{type}}(P_{s,i}, P_{s+1,j}) + \lambda f_{\text{comp}}(P_{s+1,j}). \tag{11}$$

The second cost function is

$$f_2(P_{s,i}, P_{s+1,j}) = (1-\lambda) f_{\text{type}}(P_{s,i}, P_{s+1,j}) + \lambda f_{\text{lgth}}(P_{s+1,j}). \tag{12}$$

We take one of the $2k_u - k_e$ aggregation choices according to Equations 11 or 12 in our two experiments. The cost of a whole sequence can be computed by Equations 9 or 10.

7 USING THE A* ALGORITHM

Section 3 has shown that the size of finding an optimal aggregation sequence can be exponential. That is to say, the graph G_S—our search space—can be of exponential size. In order to avoid computing the whole graph explicitly, we use the A* algorithm [21, 47]. To save time and memory, we generate a subdivision, $P_{t,i}$, only when we are going to visit it. The A* algorithm uses a clever best-first search to find a shortest path from subdivision P_{start} to subdivision P_{goal}. For $P_{t,i}$, A* considers the exact cost of a shortest path from P_{start} to $P_{t,i}$ and estimates the cost to get from $P_{t,i}$ to P_{goal}. The A* algorithm explores the nodes earlier if they are estimated to be closer to the goal. The A* algorithm can be seen as a refinement of Dijkstra's algorithm [12].

We define $g(P_{t,i})$ to be the exact cost of a shortest path from P_{start} to $P_{t,i}$ and define $h(P_{t,i})$ to be the estimated cost to get from $P_{t,i}$ to P_{goal}. Then, the (estimated) total cost at node $P_{t,i}$ is

$$F(P_{t,i}) = g(P_{t,i}) + h(P_{t,i}). \tag{13}$$

We use either g_1 (Equation 9) or g_2 (Equation 10) for $g(P_{t,i})$; accordingly, we use either h_1 (Equation 23) or h_2 (Equation 24) for $h(P_{t,i})$. If $h(P_{t,i})$ is always bounded from above by the exact cost of a shortest path from $P_{t,i}$ to P_{goal}, A* guarantees to find a shortest path from P_{start} to P_{goal}, that is, an optimal aggregation sequence. Using estimate F (Equation 13), A* is able to reduce the search space. The better the estimation part h, the more search space A* can reduce. In the following, we show how to compute estimated cost $h(P_{t,i})$.

To narrow down the search space, we set up estimation functions for type change (Section 7.1), compactness (Section 7.2), and length (Section 7.3). These functions are meant to direct A* towards the goal. Since the number of subdivisions can be exponential, we may run out of the main memory before we find an optimal solution. To handle this problem, we introduce overestimations to find a feasible solution. Overestimations are popular when people cannot find optimal solutions using A*. For example, Pohl [53] overestimated using dynamic weighting. We propose another strategy that fits our problem. We first try finding an optimal solution by A*. If we fail to find one after we have visited a predefined number, say, W of nodes of graph G_{S}, then we restart. In the retrying, we overestimate the first K steps starting at each node (see Sections 7.1, 7.2, and 7.3). We may need to increase K and retry several times until we find a feasible solution. Because we do not want to retry too many times, we define K by

$$K = 2^k - 1, \tag{14}$$

where $k \geq 0$ is the number of retryings. When $k = 0$, we have $K = 0$, which means that the first attempt of finding a solution does not use overestimation. As $K \leq n - 1$, it holds that $k \leq \log_2 n$, which means that we need to retry $\lceil \log_2 n \rceil$ times at most. Whenever overestimating ($k \geq 1$), A* cannot guarantee optimality anymore. When we are at time t, there are at most $n - t$ steps to arrive at the goal map. We define the number of practical overestimation steps as

$$K' = \min\{K, n - t\}. \tag{15}$$

7.1 Estimating the cost of type change

To find a lower bound of the cost of type change, we simply assume that every patch will be aggregated into a patch with type T_{goal}. As long as the cost of type change is a metric, this aggregation strategy indeed yields a lower bound. For subdivision $P_{t,i}$, let $(P_{t,i} = P_{t,i'_t}, P_{t+1,i'_{t+1}} \ldots, P_{n,i'_n} = P_{\text{goal}})$, be the path that always changes the type of a smallest patch to T_{goal}. Then the estimated cost of type change is

$$h_{\text{type}}(P_{t,i}) = \sum_{s=t}^{n-1} f_{\text{type}}(P_{s,i'_s}, P_{s+1,i'_{s+1}}). \tag{16}$$

As an example, for Figure 7b, we compute h_{type} according to the "aggregation sequence" of Figure 10. Note that the step from subdivision P_{2,i'_2} to subdivision P_{3,i'_3} in Figure 10 is impossible in reality because the dark patch cannot be aggregated into patch v as they are not neighbors. However, this aggregation is allowed for estimation because we may find a shortest path as long as the estimated cost is no more than the exact cost of a shortest path. When we need to overestimate, we multiply the estimated cost of the first K' steps (see Equation 15) by K (see Equation 14). As a

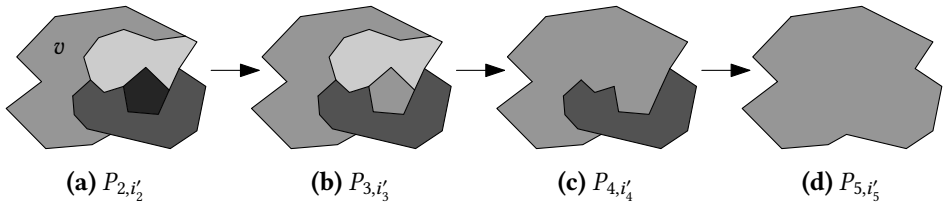


Fig. 10. An "aggregation sequence" for computing the estimated cost of type change h_{type} (see Equations 16 and 17), based on the aggregation result of Figure 7b. Note that this aggregation sequence is impossible in reality, but it is fine for estimating (see the argument in Section 7.1).

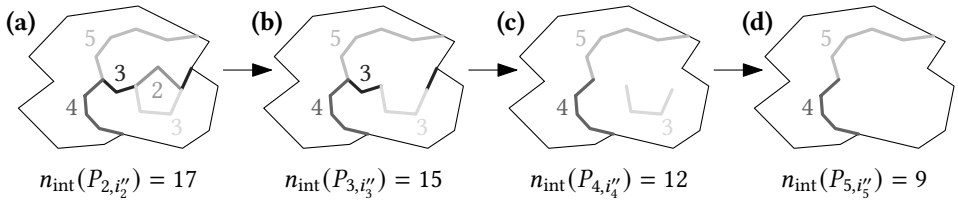


Fig. 11. An "aggregation sequence" for computing the estimated cost of compactness h_{comp} (see Equations 19 and 20), based on the number of edges. At each step we remove the boundary with the fewest edges. The numbers represent the numbers of the interior boundaries' edges. Note that this aggregation sequence is impossible in reality, but it is fine for estimating (see the argument in Section 7.1). This example corresponds to the aggregation step in Figure 7b.

result, Formula 16 is revised to

$$h_{\text{type}}(P_{t,i}) = K \sum_{s=t}^{t+K'-1} f_{\text{type}}(P_{s,i'_s}, P_{s+1,i'_{s+1}}) + \sum_{s=t+K'}^{n-1} f_{\text{type}}(P_{s,i'_s}, P_{s+1,i'_{s+1}}). \tag{17}$$

7.2 Estimating the cost of compactness

We estimate the cost of compactness based on regular polygons. The more edges a regular polygon has, the more compact it is. We assume that, at each step, we aggregate the two patches that are the least compact. Moreover, we assume that the shared boundary of the two patches has the least number of edges. We use $\mathcal{N}_{\text{ext}}$ to denote the edge number of the region's exterior boundaries. As the exterior boundaries will not be changed by aggregation, $\mathcal{N}_{\text{ext}}$ is a constant. Note that the boundary between two patches is not necessarily connected; for example, see the dark boundary with three edges in Figure 11a. For subdivision $P_{t,i}$, we denote by $B(P_{t,i})$ the set of interior boundaries and denote by $b_{\min}(P_{t,i})$ the boundary with the smallest number of edges. For our estimation, the set of interior boundaries at time $t+1$ is $B(P_{t+1,i''_{t+1}}) = B(P_{t,i}) - \{b_{\min}(P_{t,i})\}$. The estimated number of the edges for such a subdivision, $P_{t+1,i''_{t+1}}$, is

$$\mathcal{N}_{t+1,i''_{t+1}} = \mathcal{N}_{\text{ext}} + \sum_{b \in B(P_{t+1,i''_{t+1}})} \|b\|, \tag{18}$$

where notation $\|b\|$ represents the number of boundary b's edges.

From subdivision $P_{t,i}$ to subdivision $P_{t+1,i''_{t+1}}$, we get a new patch because of the aggregation. The new patch is certainly less compact than a regular polygon with $\mathcal{N}_{t+1,i''_{t+1}}$ edges. In order to estimate the compactness of the new patch, we assume that the new patch has the shape of a regular polygon with $\mathcal{N}_{t+1,i''_{t+1}}$ edges (see Equation 18). A regular polygon with $\mathcal{N}$ edges has compactness

$$c_{\text{reg}}(\mathcal{N}) = \sqrt{\frac{\pi}{\mathcal{N}} \Big/ \tan\frac{\pi}{\mathcal{N}}}.$$

Note that compactness $c_{\text{reg}}(\mathcal{N})$ increases with increasing $\mathcal{N}$. A patch with $\mathcal{N}_{t+1,i''_{t+1}}$ edges has compactness $c_{\text{reg}}(\mathcal{N}_{t+1,i''_{t+1}})$. According to our previous assumption, at each step we are always able to aggregate the two patches that are the least compact in the subdivision. We denote the compactness values of the two patches by $c_{\min 1}(P_{t,i})$ and $c_{\min 2}(P_{t,i})$. Recall that we use $C(P_{t,i})$ to denote the set of compactness values of the patches in subdivision $P_{t,i}$ (see Section 4.2). Then the set of compactness values for subdivision $P_{t+1,i''_{t+1}}$ is

$$C(P_{t+1,i''_{t+1}}) = C(P_{s,i}) \cup \{c_{\text{reg}}(\mathcal{N}_{t+1,i''_{t+1}})\} \setminus \{c_{\min 1}(P_{t,i}), c_{\min 2}(P_{t,i})\}.$$

We compute the estimated average compactness by calculating the average of the values in set $C(P_{t+1,i''_{t+1}})$. Finally, we compute the estimated cost of compactness for subdivision $P_{t+1,i''_{t+1}}$ by Equation 4.

For subdivision $P_{t,i}$, let $(P_{t,i} = P_{t,i''_t}, P_{t+1,i''_{t+1}}, \ldots, P_{n,i''_n} = P_{\text{goal}})$ be the path that always removes the two smallest compactnesses and gains a compactness of the constructed regular polygon. The estimated cost of compactness is

$$h_{\text{comp}}(P_{t,i}) = \sum_{s=t}^{n-1} f_{\text{comp}}(P_{s,i''_s}). \tag{19}$$

When overestimating, we assume that each patch in the subdivision is extremely noncompact, that is, each patch has compactness 0. One may ask if this assumption is too much. It is indeed too much for one subdivision, but it is just fine for the whole sequence as we overestimate for only a certain number of subdivisions. Based on the assumption, the cost of compactness is $f_{\text{comp}}(P_{s,i''_s}) = 1/(n-2)$, according to Equation 4. When we need to overestimate K' steps (see Equation 15), we revise the estimated cost of compactness to

$$h_{\text{comp}}(P_{t,i}) = \sum_{s=t}^{t+K'-1} \frac{1}{n-2} + \sum_{s=t+K'}^{n-1} f_{\text{comp}}(P_{s,i''_s}). \tag{20}$$

7.3 Estimating the cost of length

At time s, there are $n-s+1$ patches. There can be as few as $n-s$ interior boundaries. In order to find a lower bound for the cost of length, we keep only the necessary number, $n-s$, of shortest boundaries at each step (see Figure 12). Then, we compute the estimated cost of length according to Equation 6.

For subdivision $P_{t,i}$, let $(P_{t,i} = P_{t,i'''_t}, P_{t+1,i'''_{t+1}}, \ldots, P_{n,i'''_n} = P_{\text{goal}})$ be the path that always keeps the necessary number of shortest interior boundaries. The estimated cost of length is

$$h_{\text{lgth}}(P_{t,i}) = \sum_{s=t}^{n-1} f_{\text{lgth}}(P_{s,i'''_s}). \tag{21}$$

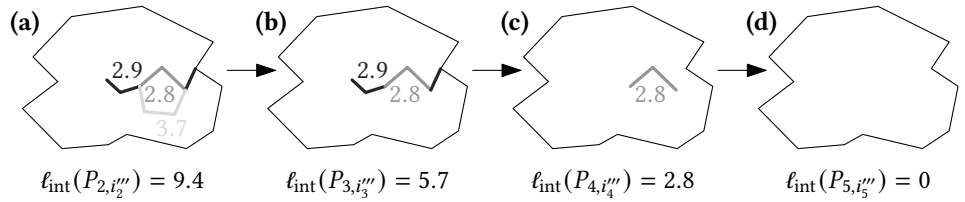


Fig. 12. An "aggregation sequence" for computing the estimated cost of length h_{lgth} (see Equations 21 and 22), based on the lengths of interior boundaries. At each step, we keep the necessary number of interior boundaries with least lengths in order to find a lower bound of the total length of the interior boundaries, i.e., $\ell_{\text{int}}(P_{s,i_s'''})$. The numbers represent the lengths of the interior boundaries. Note that this aggregation sequence is impossible in reality, but it is fine for estimating (see the argument in Section 7.1). This example corresponds to the aggregation step in Figure 7b.

When overestimating, we use the interior length of subdivision $P_{t,i}$ as the cost of length for each of the first K' steps (see Equation 15), even though we are removing interior boundaries step by step. As a result, we revise Formula 21 to

$$h_{\text{lgth}}(P_{t,i}) = \sum_{s=t}^{t+K'-1} f_{\text{lgth}}(P_{t,i}) + \sum_{s=t+K'}^{n-1} f_{\text{lgth}}(P_{s,i_s'''}). \tag{22}$$

7.4 Combining estimated costs

In accordance with our two combinatorial costs in Section 4.4, we define two estimated-cost functions:

$$h_1(P_{t,i}) = (1-\lambda)h_{\text{type}}(P_{t,i}) + \lambda h_{\text{comp}}(P_{t,i}), \tag{23}$$

and

$$h_2(P_{t,i}) = (1-\lambda)h_{\text{type}}(P_{t,i}) + \lambda h_{\text{lgth}}(P_{t,i}). \tag{24}$$

8 INTEGER LINEAR PROGRAMMING

We want to compare the A* algorithm with integer linear programming in finding optimal sequences for our aggregation problem. Since integer linear programming can handle only linear constraints, we define the compactness of a subdivision as the length of the subdivision's interior boundaries. That is, we use cost function g_2 (see Equation 10). Our basic idea is to formalize the problem of finding a shortest path as an ILP. Then we solve this ILP by minimizing the total cost. As the A* algorithm performed better in our case study, here we skip the formulation of our ILP. The formulation can be found in Appendix A.

9 CASE STUDY

We have implemented our methods based on C# (Microsoft Visual Studio 2017) and ArcObjects SDK 10.6.0. We used the IBM ILOG CPLEX Optimization Studio 12.6.3.0 to solve our ILP. Our prototype is open access on GitHub[1]. We ran our case study under 64-bit Windows 10 on Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz with 4 cores; the computer has 16 GB RAM. We measured processing time by the built-in C# class *Stopwatch*. As required by ArcObjects SDK 10.6.0, we specified

[1] https://github.com/IGNF/ContinuousGeneralisation, Accessed: Jun 18, 2018.

our program to run on the 32-bit platform. We added a post-build task about "largeaddressaware" in Microsoft Visual Studio so that we were able to use up to 3 GB of the main memory[2]. Our CPLEX version may declare an optimal solution while it is not really optimal. To fix this problem, we had to disable both primal and dual presolve reductions[3].

We tested our method on a dataset from the German topographic database ATKIS DLM 50. The dataset represents the place "Buchholz in der Nordheide" at scale 1 : 50,000. Our start map is the result of collapsing areas by Haunert [22, Chapter 6]. The start map has 5,537 polygons. Our goal map was generalized from the start map by Haunert and Wolff [27], setting the scale to 1 : 250,000 (see Figure 13). The goal map has 734 polygons, which means that there are $N = 734$ regions. The distribution of region sizes is shown in Figure 14. We used a tree-based method introduced by Rada et al. [54] to define the distances of the types; the distance is the "number of edges" that we need to travel from one node to another node in the tree of type hierarchy[4] (see Figure 15). For example, the distance from type *village* to type *fence* is 2, to type *street* is 4, and to type *farm land* is 6. In this tree, the maximum distance is 6, which means $d_{\text{type\_max}} = 6$ for Equation 1. According to Rada et al. [54], the resulting cost function for type change is a metric.

9.1 Using costs of type change and compactness

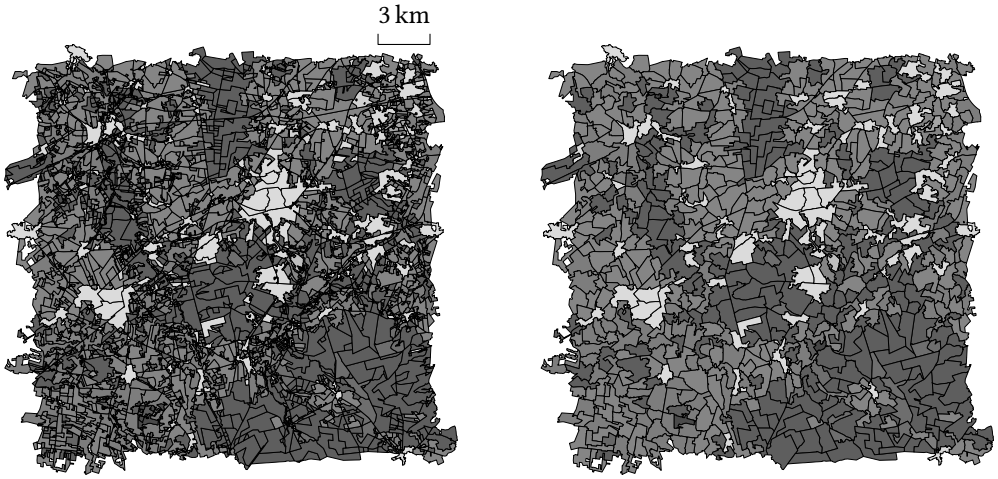
As illustrated in Section 4.4, we compare the A* algorithm and the greedy algorithm using $g_1(P_{t,i})$, which is a combination of the costs of type change and compactness (see Equation 9). For A*, we overestimated whenever we could not find a solution after having visited W nodes (see Section 7). We tried $W = 200{,}000$ and $W = 400{,}000$ (if we could use more main memory, then we could test by using some larger W). The results are shown in Table 1. Comparing to A*, the greedy algorithm visited fewer nodes and arcs in graph G_S and used much less time. However, A* managed to find solutions with lower total cost, 117.3 (or 117.2), which is 2.8% less than the total cost of the greedy algorithm, 120.7. When $W = 200{,}000$, we are sure that we have found optimal solutions for 702 of the 734 regions (95.6%), while the greedy algorithm solved only 408 (55.6%) to optimality; see column #OS in Table 1. For the other 32 regions, both algorithms have found feasible solutions (see column #FS in Table 1). Although some of the feasible solutions may also be optimal, we cannot verify that only from the cost values.

In accordance with Section 7, for region with ID i we define k_i as the least number of repetitions that we do to find a feasible solution. We define the total number of repetitions as $k_{\text{sum}} = \sum_{i=1}^{N} k_i$, where $N = 734$ is the number of the regions. After increasing W to 400,000, A* found optimal aggregation sequences for only two more regions, but k_{sum} decreased quite a bit, from 102 to 89. The numbers of regions that needed certain overestimation steps are shown in Figure 16. Besides, A* visited more arcs and nodes, used more time, but got (slightly) less cost when increasing W to 400,000. Although the number of regions that needed overestimation is relatively small, A* spent most of the running time on those few regions: 4.4% and 4.1% of the regions caused 93.2% and 95.5% of the total running time, respectively (see Table 1).

[2]The details of the setting can be found at http://stackoverflow.com/questions/2597790/can-i-set-largeaddressaware-from-within-visual-studio, Accessed: Nov 1, 2017.

[3]For more details about the problem, see http://www-01.ibm.com/support/docview.wss?uid=swg1RS02094, Accessed: Nov 12, 2017.

[4]More information about land-cover types can be found at http://www.atkis.de/dstinfo/dstinfo2.dst_gliederung, Accessed: Nov 1, 2017.



(a) Start map, 5,537 polygons, at scale 1 : 50,000

(b) Goal map, 734 polygons, at scale 1 : 250,000

2101: Village, town, city
2112: Industrial area
2114: Construction area
2201: Sport facility
2202: Leisure facility
2213: Cemetery
2230: Golf course
2301: Mining, pit, quarry
3103: Square
3302: Airport, air strip
4101: Farm land
4102: Grass land
4103: Garden land
4104: Heath
4105: Swamp
4107: Wood, forest
4108: Bosk
4109: Specialized crop
4111: Wet ground
5112: Lake, barrierlake, pond

(c) The 20 land-cover types appearing in our data

Fig. 13. The data of our case study.

The details of some regions are presented in Table 2. According to the entries with overestimation factor $K_i = 0$, we often have ratios $R_{\text{type}} = 1$ and $R_{\text{comp}} > 1$. When factor $K_i = 0$, we did not overestimate for region i. The estimated cost must be smaller or equal to the exact cost, which results in $R_{\text{type}} \geq 1$ and $R_{\text{comp}} \geq 1$. Ratio $R_{\text{type}} = 1$ means that our estimation for the cost of type change is the best. A larger R_{comp} means a poorer estimation for the cost of shape.

According to columns n and K of Table 2, $\text{A}^\star_{200,000}$ managed to find optimal solutions for all the regions with fewer than 15 polygons, and only found feasible solutions for any region with more than 21 polygons. Among the 702 regions that $\text{A}^\star_{200,000}$ solved to optimality, the greedy algorithm failed to find optimal solutions for 294 regions. Solutions of the greedy algorithm cost at most 41.7% more than solutions of $\text{A}^\star_{200,000}$; for region 85, the greedy algorithm yields a solution of cost 0.777, while the solution of $\text{A}^\star_{200,000}$ has cost 0.548 (see Figure 17). As the patches in the two sequences are the same, the two results have the same cost of compactness. The main difference is the choice of the first step, from 8 patches to 7. When aggregating the smallest patch on the start map with the surrounding patch, our greedy algorithm chooses the type which is closer to the goal type. In this

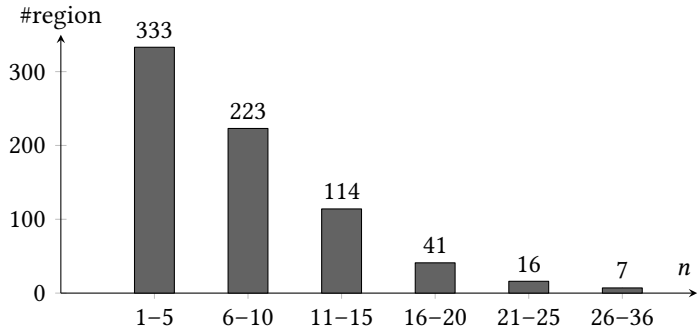


Fig. 14. Distribution of the region sizes: the y-axis shows how many regions of a given size range are contained in our dataset.

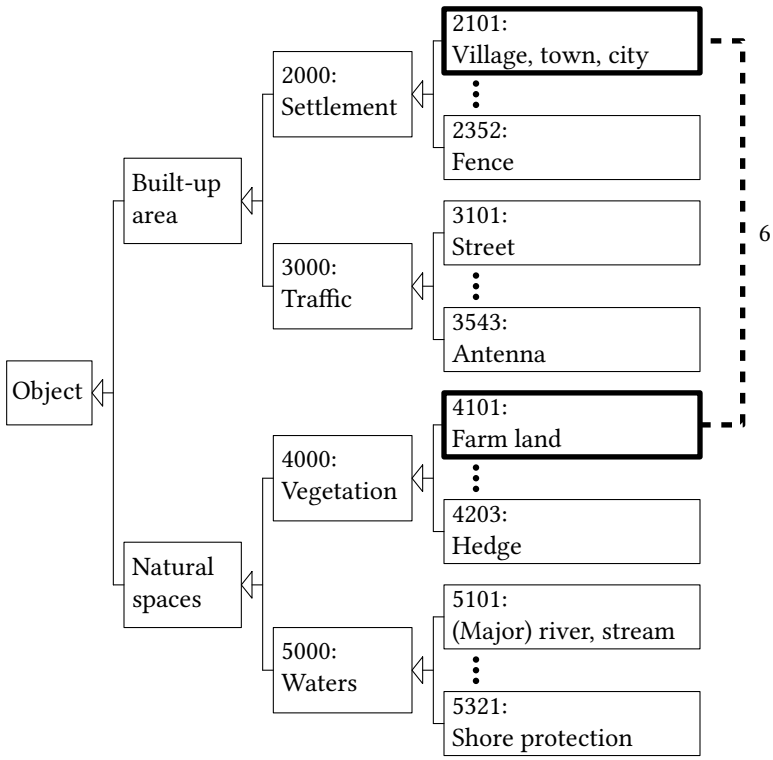


Fig. 15. The tree of type hierarchy used in our case study. For example, the distance between types *village* and *farm land* is 6.

case, the smallest patch has type 5112, and the surrounding one has 2112. The type of the goal patch is 4102. According to Figure 15, type distances $d_{\text{type}}(5112, 4102) = 4$ and $d_{\text{type}}(2112, 4102) = 6$. As a

Table 1. A comparison of the greedy algorithm and $A^\star$ when using cost function g_1 (see Equation 9). For $A^\star$, we used two settings, i.e., $W = 200{,}000$ and $W = 400{,}000$. Column #OS shows the numbers of regions that we obtained optimal solutions. Column #FS presents the numbers and the percentages of regions (out of $N = 734$) that we obtained feasible (non-optimal) solutions. Variable k_{sum} is the total number of repetitions. Columns #nodes and #arcs are the total numbers of nodes and arcs that $A^\star$ visited (for instances where we needed overestimation, only the final attempt was counted). Columns $\sum g_{\text{type}}$, $\sum g_{\text{comp}}$, and $\sum g_1$ respectively denotes the sums of $g_{\text{type}}(P_{\text{goal}})$, $g_{\text{comp}}(P_{\text{goal}})$, and $g_1(P_{\text{goal}})$ over all the 734 instances (see Equations 2, 5, and 9). The percentage in the *Time* column is the fraction of the runtime spent on solving the instances that we obtained feasible solutions. For $A^\star$, the time needed for overestimation is included.

Methods	#OS	#FS	k_{sum}	#nodes	#arcs	$\sum g_{\text{type}}$	$\sum g_{\text{comp}}$	$\sum g_1$	Time (min)
Greedy	408	326 (44.4%)		$5.5 \cdot 10^3$	$4.8 \cdot 10^3$	53.2	188.2	120.7	0.1 (74.6%)
$A^\star_{200{,}000}$	702	32 (4.4%)	102	$3.6 \cdot 10^6$	$5.7 \cdot 10^6$	51.4	183.2	117.3	51.6 (93.2%)
$A^\star_{400{,}000}$	704	30 (4.1%)	89	$6.5 \cdot 10^6$	$9.8 \cdot 10^6$	51.4	183.1	117.2	93.1 (95.5%)

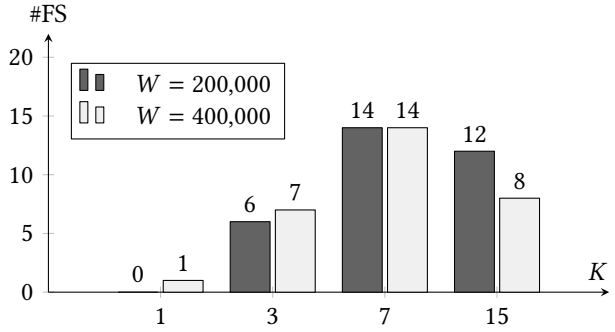


Fig. 16. The numbers of regions where $A^\star$ was forced to use the given overestimation parameters in order to find a solution without exploring more than $W \in \{200{,}000;\ 400{,}000\}$ nodes of the subdivision graph.

result, our greedy algorithm uses 5112 as the type for the new patch. This choice is a big mistake because the type of the largest patch on the start map will have to be changed twice during the aggregation. These changes cause a cost more than the sequence obtained by $A^\star_{200{,}000}$, where the largest patch on the start map is changed to the target type directly.

Among the 32 regions that $A^\star_{200{,}000}$ failed to solve optimally, the greedy algorithm outperformed $A^\star_{200{,}000}$ for 15 regions (46.9%). Among these, solutions of the greedy algorithm cost at most 15.9% less than solutions of $A^\star_{200{,}000}$; for region 543, the greedy algorithm yields a solution of cost 0.112, while the solution of $A^\star_{200{,}000}$ costs 0.134. For this instance, $A^\star_{200{,}000}$ used overestimation parameter $K = 7$ (marked in Table 2). Figure 18 shows some intermediate results obtained by $A^\star_{200{,}000}$ and the greedy algorithm. Interestingly, the two methods produced the same sequence until there were 8 patches left. Then due to the overestimation, $A^\star_{200{,}000}$ did some bad moves because the bad aggregation sequence still seemed better than other sequences. In contrast, the greedy algorithm was looking for locally good aggregations. Among the 32 regions that $A^\star_{200{,}000}$ failed to solve optimally, solutions of the greedy algorithm cost at most 17.4% more than solutions of $A^\star_{200{,}000}$; for region 155, the greedy algorithm yields a solution of cost 0.372, while the solution of $A^\star_{200{,}000}$ costs 0.317 (marked in Table 2).

Table 2. The costs in detail of some regions, where $W = 200{,}000$. Parameters n and m are the numbers of patches and adjacencies on the start map, respectively. Parameter K is the overestimation factor, defined in Section 3. We evaluate the quality of our estimations for type change and compactness by listing the numbers $R_{\text{type}} = g_{\text{type}}(P_{\text{goal}})/h_{\text{type}}(P_{\text{start}})$ and $R_{\text{comp}} = g_{\text{comp}}(P_{\text{goal}})/h_{\text{comp}}(P_{\text{start}})$. Note that if $h_{\text{type}}(P_{\text{start}}) = 0$, then we have $g_{\text{type}}(P_{\text{goal}}) = 0$; in this case, we define $R_{\text{type}} = 1$. The marked entries are discussed in the text.

ID	n	m	K	g_{type}	g_{comp}	R_{type}	R_{comp}	Time (s)
94	32	74	15	0.029	0.266	0.135	0.531	177.9
590	30	64	15	0.216	0.273	0.164	0.510	153.0
436	27	56	15	0.273	0.330	0.439	0.550	123.3
386	26	61	15	0.280	0.296	0.279	0.474	152.6
112	26	60	15	0.216	0.306	0.173	0.490	126.3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
424	20	42	7	0.339	0.292	0.623	0.746	63.9
543	20	40	7	0.000	0.267	1.000	0.681	72.7
165	20	38	7	0.102	0.355	0.347	0.903	66.2
537	19	45	7	0.525	0.328	0.702	0.791	77.4
503	19	36	7	0.199	0.246	0.525	0.595	59.9
343	19	33	7	0.164	0.355	0.586	0.857	73.1
179	22	44	3	0.355	0.308	0.967	1.716	50.8
298	22	43	3	0.176	0.268	0.948	1.471	41.1
177	22	40	3	0.046	0.276	0.853	1.578	51.8
462	18	40	3	0.130	0.239	0.682	1.155	57.3
463	17	35	3	0.234	0.238	0.799	1.087	42.0
155	15	32	3	0.324	0.310	0.878	1.243	34.5
53	21	38	0	0.047	0.315	1.000	5.160	16.8
358	21	32	0	0.044	0.337	1.000	6.264	0.6
410	20	36	0	0.135	0.334	1.000	5.553	17.3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Finally, an optimal aggregation sequence of region 53 (third-last row in Table 2) obtained by $\mathrm{A}^{\star}_{200{,}000}$ is shown in Figure 19.

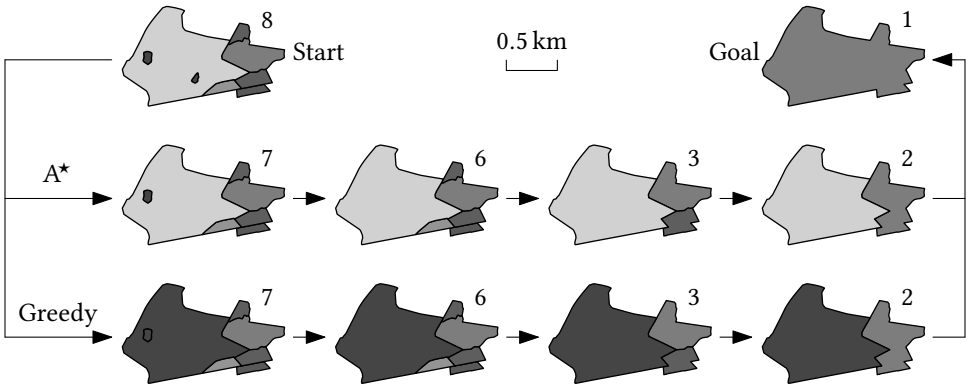


Fig. 17. Aggregation sequences of region 85 obtained by A* and the greedy algorithm. In order to save space, we did not show the results when there are 4 or 5 patches, which one can easily deduce. The numbers indicate the numbers of patches. In the sequence obtained by A*, the type of the largest polygon on the start map changed only once, which is good; while the type of the largest polygon changed twice in the sequence obtained by the greedy algorithm.

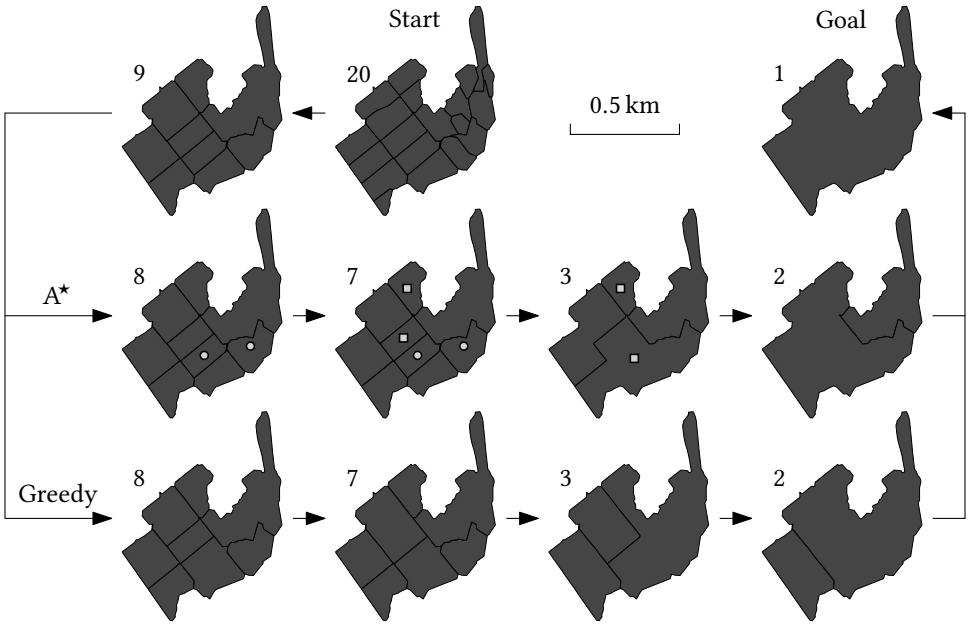


Fig. 18. Some intermediate subdivisions of region 543 obtained by A* and the greedy algorithm. In the sequence obtained by A*, a pair of circles or a pair of squares indicates that the two parts are actually in the same patch. The numbers indicate the numbers of patches.

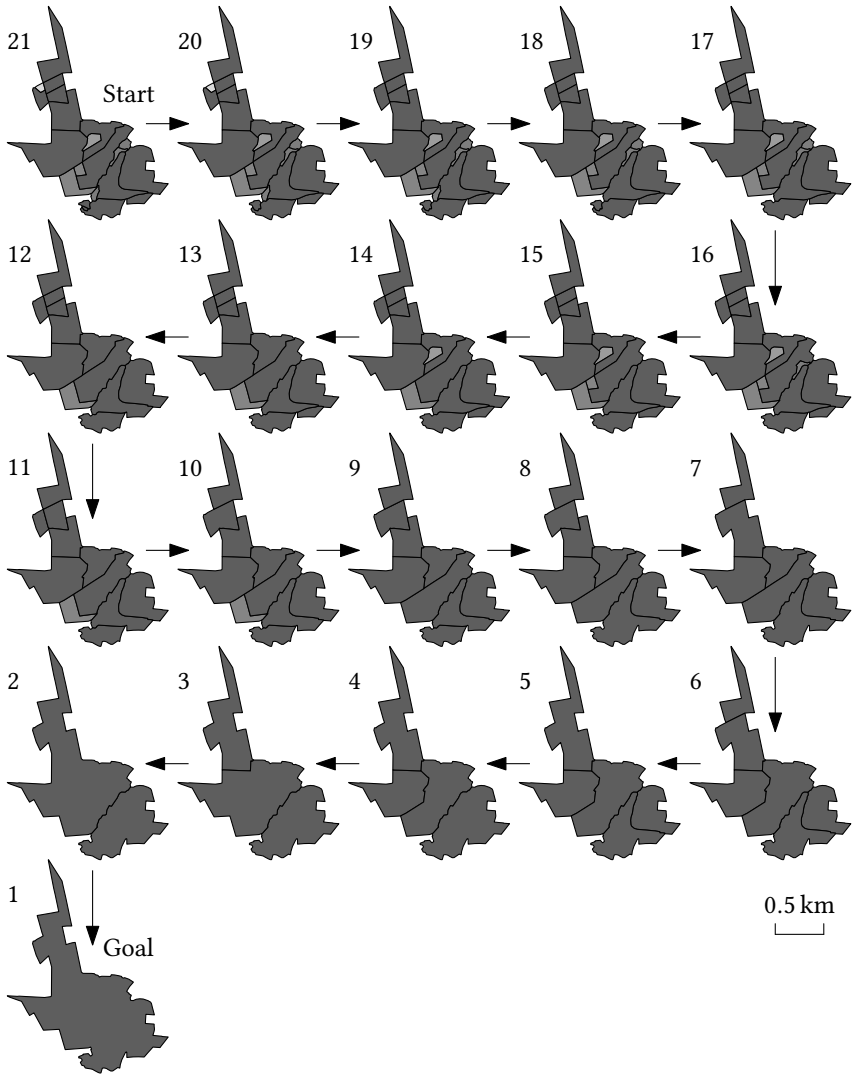


Fig. 19. An optimal sequence of intermediate subdivisions of region 53 obtained by A* using the costs of type change and compactness. The numbers indicate the numbers of patches.

Table 3. A comparison of the greedy algorithm, the A* algorithm, and the ILP-based algorithm when using cost function g_2 (see Equation 10). The notations are the same as in Table 1. Columns $\sum g_{\text{type}}$, $\sum g_{\text{lgth}}$, and $\sum g_2$ respectively represent the sums of $g_{\text{type}}(P_{\text{goal}})$, $g_{\text{lgth}}(P_{\text{goal}})$, and $g_2(P_{\text{goal}})$ over all the 734 instances (see Equations 2, 8, and 10).

Methods	#OS	#FS	k_{sum}	#nodes	#arcs	$\sum g_{\text{type}}$	$\sum g_{\text{lgth}}$	$\sum g_2$	Time (min)
Greedy	430	304 (41.4%)		$5.5 \cdot 10^3$	$4.8 \cdot 10^3$	53.0	858.5	455.8	0.1 (70.7%)
$A^\star_{200{,}000}$	695	39 (5.3%)	150	$3.7 \cdot 10^6$	$7.3 \cdot 10^6$	52.0	824.5	438.2	44.8 (92.3%)
$ILP_{100\,s}$	449	69 (9.4%)							421.5 (27.3%)
$ILP_{200\,s}$	475	57 (7.8%)							719.2 (26.4%)

9.2 Using costs of type change and length

We compare the greedy algorithm, A*, and ILP using $g_2(P_{t,i})$, a combination of the costs of type change and length (see Equation 10). For A*, we overestimated whenever we could not find a solution after having visited $W = 200{,}000$ nodes (see Section 7). The most time-consuming instance for A* was region 94, for which A* took 104.1 s (including repetitions) to find a feasible solution with overestimation factor $K = 31$. To avoid waiting too long, we set the time limit to 100 s for our ILP to run on one region. Note that the time limit included the time that our ILP used to set up the variables and constraints (see Sections A.1 and A.3). If no optimal solution was found in this time limit, a feasible solution (if found) would be returned. For some large instances, the ILP could not find any solution.

Using a similar format as in Table 1, we present the statistics in Table 3. A* found optimal solutions for 695 of the 734 regions (94.7%). Again, it spent most of the running time on the few regions that needed overestimation: 5.3% of the regions caused 92.3% of the total running time. The solutions by A* cost 438.2 in total, which is 3.9% less than 455.8, the total cost of the solutions by the greedy algorithm.

A* managed to find optimal solutions for all the regions with fewer than 15 polygons, and found only feasible solutions for the regions with more than 21 polygons. In the 39 (out of 734) regions that A* failed to solve optimally, the greedy algorithm outperformed A* for 8 regions (20.5%), which is 26.4% less comparing to the first experiment (46.9%). $ILP_{100\,s}$ managed to find optimal solutions for all the regions with fewer than 8 polygons, and failed to find optimal solutions for any region with more than 8 polygons. In none of the 39 regions that A* failed to solve optimally did $ILP_{100\,s}$ find a feasible solution. Overall, $ILP_{100\,s}$ found optimal solutions for 449 regions and found feasible solutions for 69 regions. The distributions of these regions are shown in Figures 20 and Figure 21 There are 216 regions for which our $ILP_{100\,s}$ failed to find any solution. For 22 of those regions, we did not have enough main memory to set up the variables and constraints; each of these regions has 21 polygons at least. For 67 of those regions, our $ILP_{100\,s}$ ran out of the main memory before finding any feasible solution; these regions have 14 to 20 polygons. Note that we allowed our program to use 3 GB of the main memory at most. For 123 of the 216 regions, $ILP_{100\,s}$ failed to find any solution during the time limit; these regions have 9 to 13 polygons. For the remaining 4 regions, the reason of our ILP's fail is unclear due to the fact that the solver CPLEX is like a black box for us. After we increased the time limit to 200 s for each region, $ILP_{200\,s}$ solved 475 regions to optimality, which is 26 regions more than using $ILP_{100\,s}$. Every of the 26 regions has 6 to 10 polygons. Figure 20

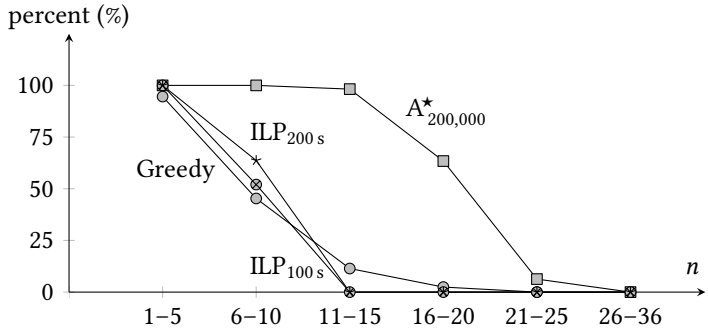


Fig. 20. The percentage of regions that were solved optimally by the greedy algorithm, $A^\star$, and our ILP. Note that the numbers of regions according to n (the number of polygons on the start map in one region) are shown in Figure 14.

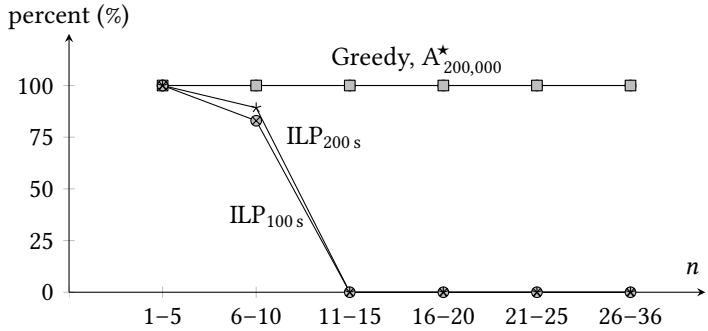


Fig. 21. The percentage of regions for which we found at least feasible solutions by the three algorithms. Note that the numbers of regions according to n (the number of polygons on the start map in one region) are shown in Figure 14.

shows the percentages of the regions that are solved optimally by the three algorithms. Figure 21 shows the percentages of the regions that the three algorithms found feasible solutions. Figure 22 shows the number of regions for which the ILP found optimal, feasible, or no solutions when using the two time limits, i.e., 100 s and 200 s.

Among all the instances that were solved to optimality by $A^\star$ in both experiments (i.e., Sections 9.1 and 9.2), region 358 (marked in Table 2) is the largest one. In both experiments, the cost of type change is 0.044. The optimal aggregation sequences for this region obtained by using costs g_1 and g_2 are shown in Figure 23. We, however, noticed some unpleasant aggregates. The step from 8 patches to 7 patches when using cost function g_1 is a bad move (see Figure 23b). Instead we expect the result of Figure 23a. Using cost function g_2, we had a similar problem. The subdivision with 7 patches is such an example, where we expect the result of Figure 23c. In an earlier version of this paper [see 52], we tried a combination of minimizing type changes and maximizing the sum of the smallest compactness values, over the whole sequence. For that objective, we had a similar problem as in Figure 23b. This problem, however, can be fixed easily by forbidding two patches to aggregate if their common boundary is too short. Moreover, there are two more possible solutions. First, we could integrate the shared length into our cost function, as did by van Oosterom [70]. Second, we

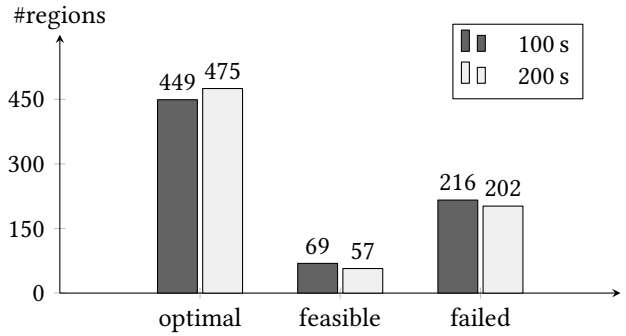


Fig. 22. The number of regions for which our ILP found optimal, feasible, or no solutions when using time limits 100 s and 200 s. Using twice the time, our ILP was able to solve only 5.8 % more instances to optimality.

could weight the cost of shape more heavily (i.e., increasing weight factor λ of Equations 9 and 10). According to our experiences, the weight factor that we applied defines a reasonable trade-off between the different conflicting objectives, when considering a solution as a whole. However, we are far from claiming that the applied weight factor has been optimally chosen. This would probably require a user study.

10 CONCLUDING REMARKS

In this paper, we investigated the problem of finding optimal sequences for area aggregation. We compared three methods to solve this problem, namely, a greedy algorithm, the A* algorithm, and an ILP-based algorithm. The greedy algorithm is used as a benchmark. Unsurprisingly, it ran faster than the other two methods by far. According to our experiments, A* found area aggregation sequences with the least total cost over all regions. For some instances, however, A* had to overestimate in order to find feasible solutions. Compared to the greedy algorithm, A* reduced the total costs by 2.8% and 3.9% in the two experiments. Although the amount is small, it is worth to use A* because optimization methods can help us to evaluate the quality of a model [25, 28, 29]. For example, Figure 23b shows that even an *optimal* sequence can be bad. If it were not for A*, we could not tell if the bad result was caused by the greedy algorithm or by the model. Because of A*, we are sure that the bad result is from our model of minimizing the type change and the compactness. The ILP-based algorithm finds optimal solutions for some regions, but for some of the other regions it cannot even find a feasible solution. Compared to the ILP-based algorithm, A* used less memory yet found optimal solutions for more regions.

Our A* has a good estimation for the cost of type change, which helps a lot to reduce the search space. Our estimation for the cost of shape (compactness or length) is poor. There are two ways to improve A* in terms of solving more instances to optimality while using the same limit of main memory. First, during the searching we can forget a node of the graph (see Figure 3) if all the neighbors of this node have been visited. By testing a case, we learned that half of the nodes can be forgotten during the pathfinding process. In this way, we can release some main memory and visit more nodes. Once we arrive at the goal, we know the cost for an optimal solution (the least cost). As many visited nodes have been forgotten, we do not have the shortest path so far. We need to run A* again. This time we know for sure that a path is not optimal if its cost, the sum of the exact cost and the estimated cost, is more than the least cost (of the optimal solution found previously). Consequently, we are able to prune some branches earlier than the first time we run A*. In this way,

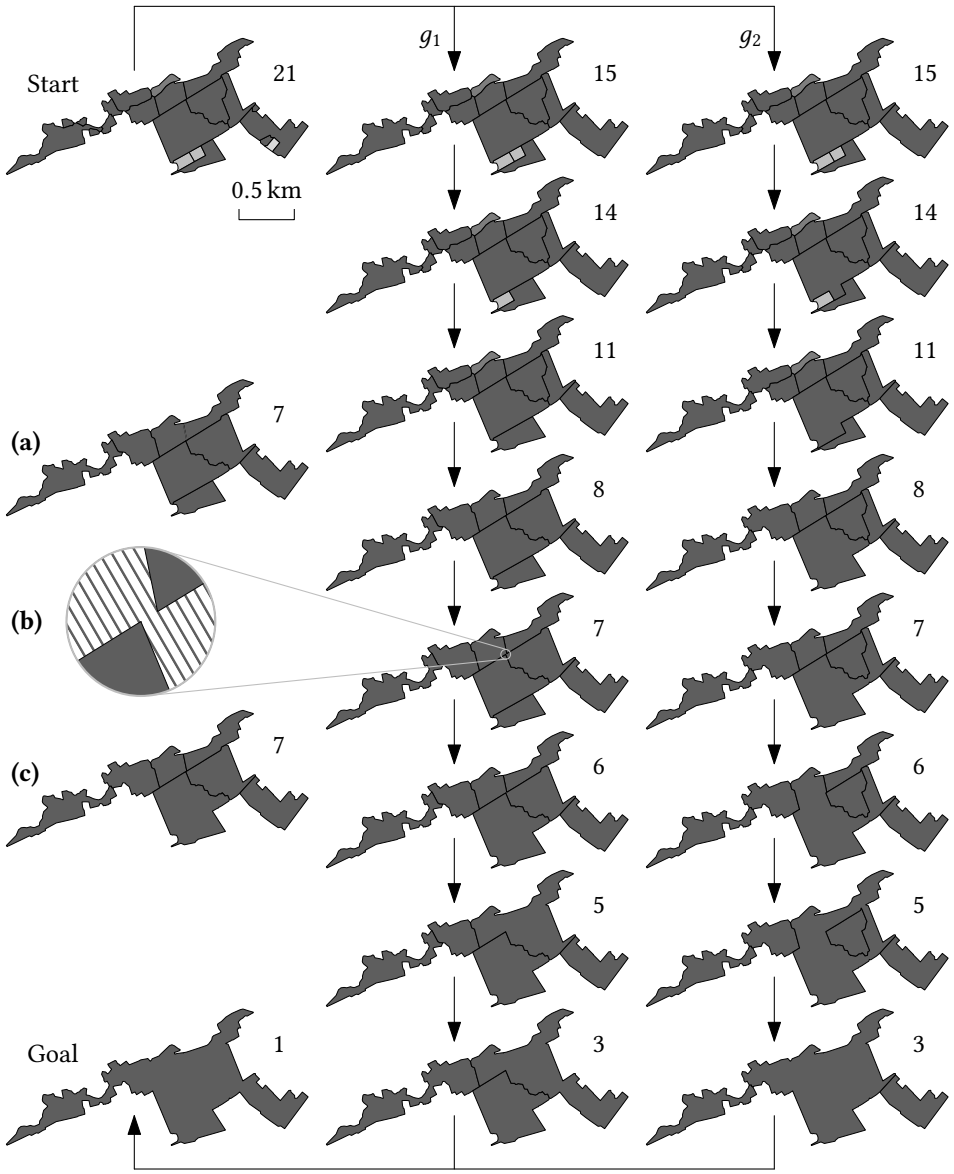


Fig. 23. Some intermediate subdivisions of region 358 obtained by A* with different cost functions. The numbers indicate the numbers of patches. The step from 8 patches to 7 patches when using cost function g_1 is a bad move; see figure (b). Instead, we expect the result of figure (a). Using cost function g_2, we had a similar problem. The subdivision with 7 patches is such an example, where we expect the result of figure (c).

we manage to save some main memory. As a result, we are more likely to find optimal solutions when the main memory is limited. Second, if we obtain a solution based on overestimation, then we know the cost of this non-optimal solution. We may decrease the overestimation factor by pruning the branches that cost more than the non-optimal solution.

We may speed up our ILP-based algorithm using a so-called cutting-plane approach as Oehrlein and Haunert [45]. Also, we can add more constraints to reduce the choices of variables. For example, assignment to a given center r is symmetric, hence we have

$$z_{t,p,q,r} = z_{t,q,p,r} \qquad \forall t \in T \setminus \{1, n\}, \forall p, q, r \in P.$$

Whether adding such kinds of constraints always speeds up our ILP is not clear because the solver, CPLEX, is a black box to us. Although integer linear programming may be not good at finding optimal sequences for area aggregation, it is relatively easy to formulate problems as ILPs. As stated by Cormen et al. [8, p. 861], "an efficient algorithm designed specifically for a problem will often be more efficient than linear programming both in theory and in practice. The real power of linear programming comes from the ability to solve new problems."

We may improve both the A* algorithm and the ILP-based algorithm by integrating the greedy algorithm. The idea is that we use the greedy algorithm to find a solution. Then we can use the cost of the solution as an upper bound to prune the branches of A* and the ILP. Once we see that the cost of a branch is larger than the upper bound, we can ignore that branch because it will not yield an optimal solution.

In cartography, there are many more requirements for area aggregation. For example, one requirement is to keep important land-cover areas for a longer time (such as a settlement surrounded by farmlands). This requirement can be achieved by incorporating the idea of Dilo et al. [13]. They gave each type a weight, then defined the importance of a patch by the product of the area size and the type weight. While in our method, we used only the area size as importance. Another requirement is that aggregating two areas may result in an area with a generalized type, as did by van Smaalen [74]. For example, aggregating *farmland* with *hedge* yields an area with type *vegetation*. In our setting, we ignored the fact that some features may inherently take linear forms (e.g., rivers). These issues can be considered in our future work.

ACKNOWLEDGMENTS

We thank Thomas C. van Dijk, Joachim Spoerhase, and Sabine Storandt for their valuable suggestions. We are grateful to Martijn Meijers for prereviewing an earlier version of this paper. We thank the anonymous reviewers for their comments, which were very helpful for us to improve the paper.

REFERENCES

[1] Bradley, S. P., Hax, A. C., and Magnanti, T. L. 1977. *Applied Mathematical Programming*. Addison-Wesley Publishing Company. http://web.mit.edu/15.053/www/AMP.htm (cit. on p. 37).

[2] Brewer, C. A. and Buttenfield, B. P. 2007. Framing guidelines for multi-scale map design using databases at multiple resolutions. *Cartography and Geographic Information Science*, 34, 1, 3–15. DOI: 10/dm5jj6 (cit. on p. 3).

[3] Burghardt, D. 2005. Controlled line smoothing by snakes. *GeoInformatica*, 9, 3, 237–252. DOI: 10/dfjwz5 (cit. on p. 6).

[4] Cecconi, A. 2003. *Integration of cartographic generalization and multi-scale databases for enhanced web mapping*. Ph.D. Dissertation. Universität Zürich, Switzerland. DOI: 10/c5kd (cit. on p. 5).

[5] Chazal, F., Lieutier, A., Rossignac, J., and Whited, B. 2010. Ball-map: Homeomorphism between compatible surfaces. *International Journal of Computational Geometry & Applications*, 20, 3, 285–306. DOI: 10/dnrwhn (cit. on p. 5).

[6] Cheng, T. and Li, Z. 2006. Toward quantitative measures for the semantic quality of polygon generalization. *Cartographica*, 41, 2, 487–499. DOI: 10.3138/0172-6733-227U-8155 (cit. on pp. 9, 10).

[7] Chimani, M., van Dijk, T. C., and Haunert, J.-H. 2014. How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In *Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS)*, 243–252. DOI: 10.1145/2666310.2666414 (cit. on p. 6).

[8] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2009. *Introduction to Algorithms*. (3rd ed.). The MIT Press. https://mitpress.mit.edu/books/introduction-algorithms-third-edition (cit. on pp. 7, 9, 30).

[9] Cova, T. J. and Church, R. L. 2000. Contiguity constraints for single-region site search problems. *Geographical Analysis*, 32, 4, 306–329. DOI: 10.1111/j.1538-4632.2000.tb00430.x (cit. on p. 41).

[10] Danciger, J., Devadoss, S. L., Mugno, J., Sheehy, D., and Ward, R. 2009. Shape deformation in continuous map generalization. *GeoInformatica*, 13, 2, 203–221. DOI: 10/d24vxs (cit. on p. 5).

[11] Deng, M. and Peng, D. 2015. Morphing linear features based on their entire structures. *Transactions in GIS*, 19, 5, 653–677. DOI: 10.1111/tgis.12111 (cit. on p. 5).

[12] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1, 269–271. DOI: 10/dpvk8c (cit. on pp. 9, 14).

[13] Dilo, A., van Oosterom, P., and Hofman, A. 2009. Constrained tGAP for generalization between scales: The case of Dutch topographic data. *Computers, Environment and Urban Systems*, 33, 5, 388–402. DOI: 10.1016/j.compenvurbsys.2009.07.006 (cit. on pp. 10, 30).

[14] Douglas, D. H. and Peucker, T. K. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10, 2, 112–122. DOI: 10.3138/fm57-6770-u75u-7727 (cit. on p. 4).

[15] Duchêne, C. et al. 2014. Generalisation in practice within national mapping agencies. In *Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation*. Lecture Notes in Geoinformation and Cartography. D. Burghardt, C. Duchêne, and W. Mackaness, (Eds.) Chap. 11, 329–391. DOI: 10.1007/978-3-319-00203-3_11 (cit. on p. 2).

[16] Frolov, Y. S. 1975. Measuring the shape of geographical phenomena: a history of the issue. *Soviet Geography*, 16, 10, 676–687. DOI: 10.1080/00385417.1975.10640104 (cit. on p. 10).

[17] Funke, S., Mendel, T., Miller, A., Storandt, S., and Wiebe, M. 2017. Map simplification with topology constraints: exactly and in practice. In *Proc. 19th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 185–196. DOI: 10/c3s3 (cit. on p. 6).

[18] Girres, J.-F. and Touya, G. 2014. Cartographic generalisation aware of multiple representations. In *Proc. 8th International Conference on Geographic Information Science (GIScience)*. M. Duckham, K. Stewart, and E. Pebesma, (Eds.) Poster (cit. on p. 5).

[19] Harrie, L. 1999. The constraint method for solving spatial conflicts in cartographic generalization. *Cartography and Geographic Information Science*, 26, 1, 55–69. DOI: 10.1559/152304099782424884 (cit. on p. 6).

[20] Harrie, L., Stigmar, H., and Djordjevic, M. 2015. Analytical estimation of map readability. *ISPRS International Journal of Geo-Information*, 4, 2, 418–446. DOI: 10.3390/ijgi4020418 (cit. on p. 10).

[21] Hart, P. E., Nilsson, N. J., and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, 4, 2, 100–107. DOI: 10.1109/TSSC.1968.300136 (cit. on p. 14).

[22] Haunert, J.-H. 2009. *Aggregation in map generalization by combinatorial optimization*. Ph.D. Dissertation. Leibniz Universität Hannover, Germany. https://dgk.badw.de/fileadmin/user_upload/Files/DGK/docs/c-626.pdf (cit. on p. 19).

[23] Haunert, J.-H., Dilo, A., and van Oosterom, P. 2009. Constrained set-up of the tGAP structure for progressive vector data transfer. *Computers and Geosciences*, 35, 11, 2191–2203. DOI: 10.1016/j.cageo.2008.11.002 (cit. on p. 3).

[24] Haunert, J.-H. and Meulemans, W. 2016. Partitioning polygons via graph augmentation. In *Proc. 9th International Conference on Geographic Information Science (GIScience)* (Lecture Notes in Computer Science). A. J. Miller, D. O'Sullivan, and N. Wiegand, (Eds.) Vol. 9927, 18–33. DOI: 10.1007/978-3-319-45738-3_2 (cit. on p. 6).

[25] Haunert, J.-H. and Sester, M. 2008. Assuring logical consistency and semantic accuracy in map generalization. *Photogrammetrie Fernerkundung Geoinformation*, 2008, 3, 165–173. https://www.dgpf.de/pfg/2008/pfg2008_3_Haunert.pdf (cit. on pp. 5, 28).

[26] Haunert, J.-H. and Wolff, A. 2010. Optimal and topologically safe simplification of building footprints. In *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS)*. A. E. Abbadi, D. Agrawal, M. Mokbel, and P. Zhang, (Eds.), 192–201. DOI: 10.1145/1869790.1869819 (cit. on p. 6).

[27] Haunert, J.-H. and Wolff, A. 2010. Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographical Information Science*, 24, 12, 1871–1897. DOI: 10/c8v8s2 (cit. on pp. 3, 6, 10, 19).

[28] Haunert, J.-H. and Wolff, A. 2016. Räumliche Analyse durch kombinatorische Optimierung. In *Handbuch der Geodäsie (6 Bände)*. Springer Reference Naturwissenschaften. W. Freeden and R. Rummel, (Eds.), 1–39. DOI: 10.1007/978-3-662-46900-2_69-2 (cit. on pp. 5, 28).

[29] Haunert, J.-H. and Wolff, A. 2017. Beyond maximum independent set: an extended integer programming formulation for point labeling. *ISPRS International Journal of Geo-Information*, 6, 11. DOI: 10.3390/ijgi6110342 (cit. on pp. 5, 28, 36).

[30] Huang, L., Ai, T., van Oosterom, P., Yan, X., and Yang, M. 2017. A matrix-based structure for vario-scale vector representation over a wide range of map scales: the case of river network data. *ISPRS International Journal of Geo-Information*, 6, 7. DOI: 10.3390/ijgi6070218 (cit. on p. 5).

[31] Jaakkola, O. 1998. Multi-scale categorical data bases with automatic generalization transformations based on map algebra. *Cartography and Geographic Information Systems*, 25, 4, 195–207. DOI: 10.1559/152304098782383016 (cit. on p. 2).

[32] Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4, 4, 373–395. DOI: 10/czqmxn (cit. on p. 36).

[33] Keane, M. 1975. The size of the region-building problem. *Environment and Planning A: Economy and Space*, 7, 5, 575–577. DOI: 10.1068/a070575 (cit. on p. 8).

[34] Li, J., Ai, T., Liu, P., and Yang, M. 2017. Continuous scale transformations of linear features using simulated annealing-based morphing. *ISPRS International Journal of Geo-Information*, 6, 8. DOI: 10.3390/ijgi6080242 (cit. on p. 5).

[35] Li, J., Li, X., and Xie, T. 2017. Morphing of building footprints using a turning angle function. *ISPRS International Journal of Geo-Information*, 6, 6. DOI: 10.3390/ijgi6060173 (cit. on p. 5).

[36] Li, W., Goodchild, M. F., and Church, R. 2013. An efficient measure of compactness for two-dimensional shapes and its application in regionalization problems. *International Journal of Geographical Information Science*, 27, 6, 1227–1250. DOI: 10/c5kg (cit. on p. 10).

[37] Li, Z. and Zhou, Q. 2012. Integration of linear and areal hierarchies for continuous multi-scale representation of road networks. *International Journal of Geographical Information Science*, 26, 5, 855–880. DOI: 10.1080/13658816.2011.616861 (cit. on p. 4).

[38] Maceachren, A. M. 1985. Compactness of geographic shape: Comparison and evaluation of measures. *Geografiska Annaler: Series B, Human Geography*, 67, 1, 53–67. DOI: 10/c329 (cit. on p. 10).

[39] Mackaness, W. A., Burghardt, D., and Duchêne, C. 2016. Map generalization. In *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, 1–16. DOI: 10/cx89 (cit. on p. 2).

[40] Meijers, M., Savino, S., and van Oosterom, P. 2016. SplitArea: An algorithm for weighted splitting of faces in the context of a planar partition. *International Journal of Geographical Information Science*, 30, 8, 1522–1551. DOI: 10.1080/13658816.2016.1140770 (cit. on p. 4).

[41] Midtbø, T. and Nordvik, T. 2007. Effects of animations in zooming and panning operations on web maps: a web-based experiment. *The Cartographic Journal*, 44, 4, 292–303. DOI: 10/dgnjmj (cit. on p. 2).

[42] Minas, J. P. and Hearne, J. W. 2016. An optimization model for aggregation of prescribed burn units. *TOP*, 24, 1, 180–195. DOI: 10.1007/s11750-015-0383-y (cit. on p. 10).

[43] Müller, J.-C., Weibel, R., Lagrange, J.-P., and Salgé, F. 1995. Generalization: State of the art and issues. In *GIS and Generalization: Methodology and Practice*. Number 1 in GISDATA. J.-C. Müller, J.-P. Lagrange, and R. Weibel, (Eds.) Chap. 1, 3–17 (cit. on p. 1).

[44] Nöllenburg, M., Merrick, D., Wolff, A., and Benkert, M. 2008. Morphing polylines: A step towards continuous generalization. *Computers, Environment and Urban Systems*, 32, 4, 248–260. DOI: 10/c7fgrw (cit. on pp. 5, 6).

[45] Oehrlein, J. and Haunert, J.-H. 2017. A cutting-plane method for contiguity-constrained spatial aggregation. *Journal of Spatial Information Science*, 15, 89–120. DOI: 10.5311/JOSIS.2017.15.379 (cit. on pp. 3, 6, 30, 41).

[46] Pantazis, D., Karathanasis, B., Kassoli, M., Koukofikis, A., and Stratakis, P. 2009. Morphing techniques: Towards new methods for raster based cartographic generalization. In *Proc. 24th International Cartographic Conference (ICC)*. https://icaci.org/files/documents/ICC_proceedings/ICC2009/html/refer/19_5.pdf (cit. on p. 2).

[47] Patel, A. Amit's A* pages. Accessed: Jul 26, 2018, (). http://theory.stanford.edu/~amitp/GameProgramming (cit. on p. 14).

[48] Peng, D., Deng, M., and Zhao, B. 2012. Multi-scale transformation of river networks based on morphing technology. *Journal of Remote Sensing*, 16, 5, 953–968. http://www.jors.cn/jrs/ch/reader/view_abstract.aspx?file_no=r11272&flag=1 (cit. on p. 5).

[49] Peng, D., Haunert, J.-H., Wolff, A., and Hurter, C. 2013. Morphing polylines based on least squares adjustment. In *Proc. 16th ICA Workshop on Generalisation and Multiple Representation (ICAGM)*. https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2013/genemappro2013_submission_6.pdf (cit. on pp. 5, 6).

[50] Peng, D. and Touya, G. 2017. Continuously generalizing buildings to built-up areas by aggregating and growing. In *Proc. 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics (UrbanGIS)*. DOI: 10.1145/3152178.3152188 (cit. on p. 5).

[51] Peng, D., Wolff, A., and Haunert, J.-H. 2016. Continuous generalization of administrative boundaries based on compatible triangulations. In *Proc. 19th AGILE Conference on Geographic Information Science, Geospatial Data in a Changing World* (Lecture Notes in Geoinformation and Cartography). T. Sarjakoski, Y. M. Santos, and T. L. Sarjakoski, (Eds.), 399–415. DOI: 10/c5kh (cit. on p. 5).

[52] Peng, D., Wolff, A., and Haunert, J.-H. 2017. Using the A* algorithm to find optimal sequences for area aggregation. In *Proc. 28th International Cartographic Conference (ICC), Advances in Cartography and GIScience* (Lecture Notes in Geoinformation and Cartography). M. P. Peterson, (Ed.), 389–404. DOI: 10.1007/978-3-319-57336-6_27 (cit. on p. 27).

[53] Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proc. 3rd International Joint Conference on Artificial Intelligence (IJCAI)*, 12–17. https://exhibits.stanford.edu/feigenbaum/catalog/sq127cx4634 (cit. on p. 15).

[54] Rada, R., Mili, H., Bicknell, E., and Blettner, M. 1989. Development and application of a metric on semantic nets. *IEEE Transactions On Systems Man And Cybernetics*, 19, 1, 17–30. DOI: 10.1109/21.24528 (cit. on p. 19).

[55] Regnauld, N. 2001. Contextual building typification in automated map generalization. *Algorithmica*, 30, 2, 312–333. DOI: 10.1007/s00453-001-0008-8 (cit. on p. 6).

[56] Saalfeld, A. 1999. Topologically consistent line simplification with the Douglas–Peucker algorithm. *Cartography and Geographic Information Science*, 26, 1, 7–18. DOI: 10/drcc5h (cit. on p. 4).

[57] Schwartges, N., Allerkamp, D., Haunert, J.-H., and Wolff, A. 2013. Optimizing active ranges for point selection in dynamic maps. In *Proc. 16th ICA Workshop on Generalisation and Multiple Representation (ICAGM)*. https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2013/genemappro2013_submission_5.pdf (cit. on p. 6).

[58] Sester, M. 2005. Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science*, 19, 8–9, 871–897. DOI: 10.1080/13658810500161179 (cit. on p. 6).

[59] Sester, M. and Brenner, C. 2005. Continuous generalization for visualization on small mobile devices. In *Proc. 11th International Symposium on Spatial Data Handling (SDH)*. P. Fisher, (Ed.), 355–368. DOI: 10.1007/3-540-26772-7_27 (cit. on p. 4).

[60] Shirabe, T. 2005. A model of contiguity for spatial unit allocation. *Geographical Analysis*, 37, 1, 2–16. DOI: 10.1111/j.1538-4632.2005.00605.x (cit. on p. 41).

[61] Smith, G., Beare, M., Boyd, M., Downs, T., Gregory, M., Morton, D., Brown, N., and Thomson, A. 2007. UK land cover map production through the generalisation of OS MasterMap. *The Cartographic Journal*, 44, 3, 276–283. DOI: 10.1179/000870407X241827 (cit. on p. 4).

[62] Stoter, J., van Smaalen, J., Bakker, N., and Hardy, P. 2009. Specifying map requirements for automated generalization of topographic data. *The Cartographic Journal*, 46, 3, 214–227. DOI: 10/fttg54 (cit. on p. 5).

[63] Šuba, R., Meijers, M., and van Oosterom, P. 2016. Continuous road network generalization throughout all scales. *ISPRS International Journal of Geo-Information*, 5, 8. DOI: 10.3390/ijgi5080145 (cit. on pp. 2, 5).

[64] Thiemann, F. and Sester, M. 2018. An automatic approach for generalization of land-cover data from topographic data. In *Trends in Spatial Analysis and Modelling: Decision-Support and Planning Strategies*. Geotechnologies and the Environment. Vol. 19. M. Behnisch and G. Meinel, (Eds.) Chap. 10, 193–207. DOI: 10/c5kj (cit. on p. 4).

[65] Timpf, S. 1998. *Hierarchical Structures in Map Series*. Ph.D. Dissertation. Technical University Vienna, Austria. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.4561&rep=rep1&type=pdf (cit. on p. 3).

[66] Tong, X., Jin, Y., Li, L., and Ai, T. 2015. Area-preservation simplification of polygonal boundaries by the use of the structured total least squares method with constraints. *Transactions in GIS*, 19, 5, 780–799. DOI: 10.1111/tgis.12130 (cit. on p. 6).

[67] Touya, G. and Dumont, M. 2017. Progressive block graying and landmarks enhancing as intermediate representations between buildings and urban areas. In *Proc. 20th ICA Workshop on Generalisation and Multiple Representation (ICAGM)*. https://kartographie.geo.tu-dresden.de/downloads/ica-gen/workshop2017/genemr2017_paper_1.pdf (cit. on p. 5).

[68] Touya, G. and Girres, J.-F. 2013. ScaleMaster 2.0: A ScaleMaster extension to monitor automatic multi-scales generalizations. *Cartography and Geographic Information Science*, 40, 3, 192–200. DOI: 10.1080/15230406.2013.809233 (cit. on p. 3).

[69] van Kreveld, M. 2001. Smooth generalization for continuous zooming. In *Proc. 5th ICA Workshop on Generalisation and Multiple Representation (ICAGM)*. http://www.staff.science.uu.nl/~kreve101/papers/smooth.pdf (cit. on p. 4).

[70] van Oosterom, P. 2005. Variable-scale topological data structures suitable for progressive data transfer: The GAP-face tree and GAP-edge forest. *Cartography and Geographic Information Science*, 32, 4, 331–346. DOI: 10/chr7sf (cit. on pp. 3, 14, 27).

[71] van Oosterom, P. and Meijers, M. 2014. Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *International Journal of Geographical Information Science*, 28, 3, 455–478. DOI: 10.1080/13658816.2013.809724 (cit. on pp. 3, 5).

[72] van Oosterom, P., Meijers, M., Stoter, J., and Šuba, R. 2014. Data structures for continuous generalisation: tGAP and SSC. In *Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation*. Lecture Notes in Geoinformation and Cartography. D. Burghardt, C. Duchêne, and W. Mackaness, (Eds.) Chap. 4, 83–117. DOI: 10.1007/978-3-319-00203-3_4 (cit. on p. 3).

[73] van Oosterom, P. and Schenkelaars, V. 1995. The development of an interactive multi-scale GIS. *International Journal of Geographical Information Systems*, 9, 5, 489–507. DOI: 10/fgzjvb (cit. on p. 4).

[74] van Smaalen, J. 2003. *Automated Aggregation of Geographic Objects*. Ph.D. Dissertation. Wageningen University (cit. on pp. 3, 30).

[75] Weibel, R. 1997. Generalization of spatial data: Principles and selected algorithms. In *Algorithmic Foundations of Geographic Information Systems*. Lecture Notes in Computer Science. Vol. 1340. M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, (Eds.) Chap. 5, 99–152. DOI: 10.1007/3-540-63818-0_5 (cit. on pp. 1, 2).

[76] Weibel, R. and Burghardt, D. 2017. Generalization, on-the-fly. In *Encyclopedia of GIS*. (2nd ed.). S. Shekhar, H. Xiong, and X. Zhou, (Eds.), 657–663. DOI: 10.1007/978-3-319-17885-1_450 (cit. on p. 2).

[77] Whited, B. and Rossignac, J. 2011. Ball-morph: Definition, implementation, and comparative evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 17, 6, 757–769. DOI: 10.1109/TVCG.2010.115 (cit. on p. 5).

[78] Williams, H. P. 2009. *Logic and Integer Programming*. (1st ed.). Springer. DOI: 10.1007/978-0-387-92280-5 (cit. on p. 37).

[79] Williams, J. C. 2002. A zero-one programming model for contiguous land acquisition. *Geographical Analysis*, 34, 4, 330–349. DOI: 10.1111/j.1538-4632.2002.tb01093.x (cit. on p. 41).

[80] Wright, J., Revelle, C., and Cohon, J. 1983. A multiobjective integer programming model for the land acquisition problem. *Regional Science and Urban Economics*, 13, 1, 31–53. DOI: 10/dqgvs5 (cit. on p. 10).

[81] Wu, S.-T., da Silva, A. C. G., and Márquez, M. R. G. 2004. The Douglas–Peucker algorithm: Sufficiency conditions for non-self-intersections. *Journal of the Brazilian Computer Society*, 9, 3, 67–84. DOI: 10/cxwv (cit. on p. 4).

[82] Young, H. P. 1988. Measuring the compactness of legislative districts. *Legislative Studies Quarterly*, 13, 1, 105–115. DOI: 10.2307/439947 (cit. on p. 10).

[83] Zoltners, A. A. and Sinha, P. 1983. Sales territory alignment: a review and model. *Management Science*, 29, 11, 1237–1256. DOI: 10.1287/mnsc.29.11.1237 (cit. on p. 41).

A FORMULATION OF INTEGER LINEAR PROGRAMMING

Linear programming is a method to optimize a *linear objective* subject to a set of *linear constraints* with some *variables*. For example, suppose that we are selling coffee. We have 3.5 kg of coffee powder and 10 kg of water. We mix the powder and the water to provide two kinds coffee with different intensities in terms of mass: 40% and 20%. The profits of the two kinds of coffee are respectively 5 € and 4 €. Our aim is to maximize the total profit of selling coffee. If we offer respectively x kg and y kg of the two kinds of coffee, then x and y are our variables. Our objective is to

$$\text{maximize} \quad 5x + 4y.$$

To provide x kg of coffee with intensity 40%, we need to use $0.4x$ kg of coffee powder and $0.6x$ kg water. Analogously, it consumes $0.2y$ kg of coffee powder and $0.8y$ kg of water to produce y kg of coffee with intensity 20%. As a result, we have four constraints:

$$\begin{aligned} 0.4x + 0.2y &\leq 3.5, \\ 0.6x + 0.8y &\leq 10, \text{ and} \\ x, y &\geq 0. \end{aligned}$$

With the objective and the constraints, we have set up a *linear program* (LP). We observed that all the feasible solutions, i.e., pairs of (x, y), fall in the gray area of Figure 24a. Drawing a line with slope $-\frac{5}{4}$, we see that every pair of (x, y) lying on the line yields the same result for $5x + 4y$, the profit we want to maximize. For example, every pair of (x, y) lying on the dashed line in Figure 24a yields profit 40 €. If we move the dashed line to the upper right, then we are able to achieve a larger value for $5x + 4y$. In order to maximize the profit, we move the dashed line to the upper right as much as possible and, at the same time, make sure that it still intersects with the gray area. Note that if the dashed line does not intersect with the gray area, then there is no feasible pair of (x, y) on the dashed line anymore. As a result, we get the optimal solution when the dashed line hits point A, where the profit is $5 \cdot 4 + 4 \cdot 9.5 = 58$ €. Karmarkar [32] proved that an LP can be solved in polynomial time.

Now we change our problem a bit. We wish to sell coffee in jugs, where each jug contains exactly 1 kg of coffee with intensity 40% or 20%. Our question becomes how many jugs of each kind of coffee we should sell in order to maximize the profit. If we sell the two kinds of coffee respectively x' and y' jugs, then the problem becomes:

$$\begin{aligned} \text{maximize} \quad & 5x' + 4y' \\ \text{subject to} \quad & 0.4x' + 0.2y' \leq 3.5, \\ & 0.6x' + 0.8y' \leq 10, \\ & x', y' \geq 0, \\ \text{and} \quad & x', y' \in \mathbb{Z}. \end{aligned}$$

For this problem, only the pairs of (x', y') represented by the gray dots of Figure 24b are feasible solutions (point A is no longer a feasible solution in this case). In order to maximize our profit, we should move the dashed line to the upper right as much as possible and, at the same time, make sure that it hits at least one of the gray dots. To solve such a problem is known as *integer linear programming*, which is NP-complete. Despite the fact, there are mathematical solvers yielding optimal solutions for some NP-complete problems in reasonable time [29]. By using these solvers, we benefit from every improvement, by their producers, for the same class of problems [29]. The

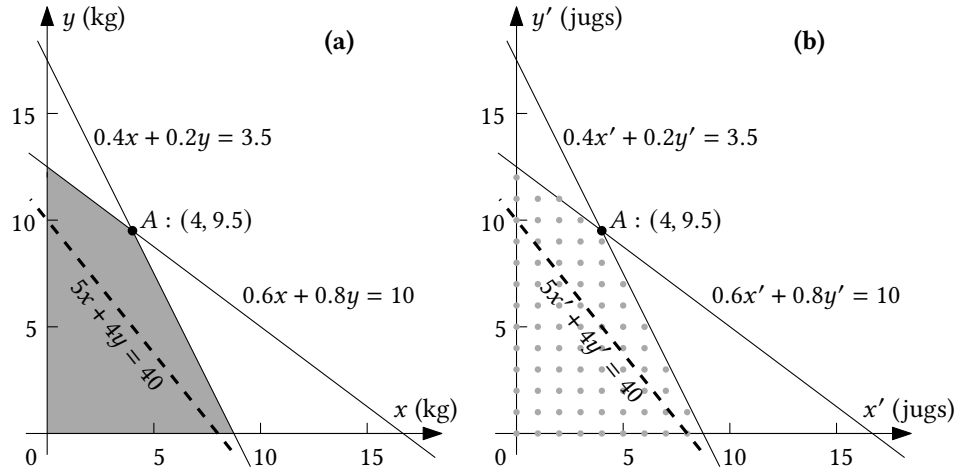


Fig. 24. Examples of linear programming (a) and integer linear programming (b). In (a), any point in the gray area is a feasible solution; in (b), only the gray points are feasible solutions.

general form of an *integer linear program* (ILP) is

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{C}^{\mathrm{T}}\boldsymbol{X} \\ \text{subject to} \quad & \boldsymbol{E}\boldsymbol{X} \leq \boldsymbol{H}, \\ & \boldsymbol{X} \geq \boldsymbol{0}, \\ \text{and} \quad & \boldsymbol{X} \in \mathbb{Z}^I, \end{aligned}$$

where vector $\boldsymbol{X}$ represents integer variables, vector $\boldsymbol{C} \in \mathbb{R}^I$, vector $\boldsymbol{H} \in \mathbb{R}^J$, and $\boldsymbol{E}$ is a $(J \times I)$-matrix over the reals. Furthermore, if we require

$$\boldsymbol{X} \in \{0, 1\}^I,$$

then we have only binary variables for an ILP. Binary variables are important because they occur regularly in optimizations [1, Section 9.2]. Also, an ILP with general (bounded) integer variables can always be translated to an ILP with binary variables [78, Section 2.3]. We are going to use binary variables in our ILP because it is more intuitive to model our problem using binary variables than using other integers.

We want to compare the A* algorithm with integer linear programming in finding optimal sequences for our aggregation problem. Since integer linear programming can handle only linear constraints, we define the compactness of a subdivision as the length of the subdivision's interior boundaries. That is, we use cost function g_2 (see Equation 10). Our basic idea is to formalize the problem of finding a shortest path as an ILP. Then we solve this ILP by minimizing the total cost. We define the *center* of a patch as the polygon to which other polygons in the same patch are assigned. At the beginning, every patch consists of only one polygon, and this polygon is the center of the patch. When we aggregate patch u into patch v, all the polygons of u are assigned to the center of v, and the type of u's polygons are changed to the type of v's center. In the following, we show how to formalize our problem as an ILP. For simplicity, we sometimes denote by *patch r* the patch using polygon r as the center at time t.

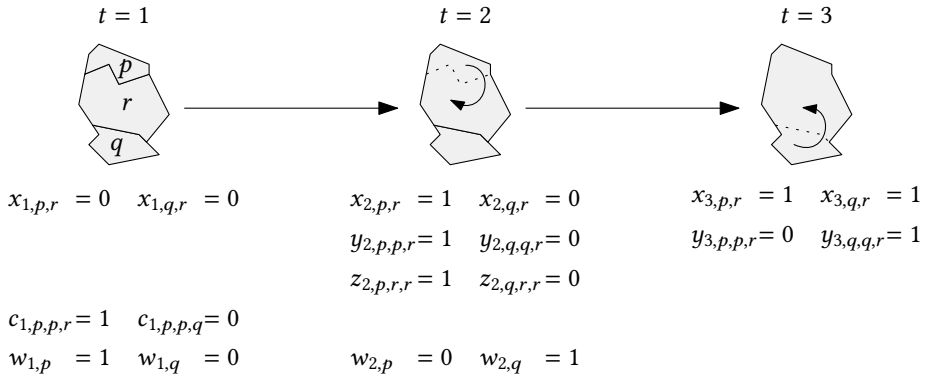


Fig. 25. Some examples of the five sets of variables for our ILP, x, y, z, c, and w. The arrows with curly arms show the aggregation steps, and the dotted lines represent the removed boundaries by the aggregation steps. There are some blank spaces in the rows of the variables because there is no corresponding variable at the specific times.

A.1 Variables

Our problem is to decide centers for polygons to be assigned. Each question of type "Is polygon p assigned to center r?" can be answered with "yes" or "no". Hence, we use binary (0–1) variables. We need five sets of variables in order to formulate our pathfinding problem as an ILP. Recall that We use $T = \{1, 2, \ldots, n\}$ to represent the set of times and use P to denote the set of n polygons on the start map (see Section 3). The first set of variables is used to tell the program our rules for area aggregation. We introduce the variable

$$x_{t,p,r} \in \{0, 1\} \qquad \forall t \in T, \forall p, r \in P$$

with the intended meaning $x_{t,p,r} = 1$ if and only if polygon p is assigned to polygon r at time t (see Figure 25 for some examples). If a polygon is a center at time t, then the polygon must be assigned to itself, that is, $x_{t,r,r} = 1$.

We use the second set of variables in order to compute the cost of type change. We introduce

$$y_{t,p,o,r} \in \{0, 1\} \qquad \forall t \in T \setminus \{1\}, \forall p, o, r \in P$$

with the intended meaning $y_{t,p,o,r} = 1$ if and only if polygon p is assigned to center o at time $t - 1$ and assigned to center r at time t (see Figure 25). Specifically, case $y_{t,p,o,o} = 1$ means that polygon p is assigned to the same center at times $t - 1$ and t.

We need a third set of variables for computing the cost of length. We introduce

$$z_{t,p,q,r} \in \{0, 1\} \qquad \forall t \in T \setminus \{1, n\}, \forall p, q, r \in P$$

with the intended meaning $z_{t,p,q,r} = 1$ if and only if polygons p and q are both assigned to center r at time t (p and q are in the same patch). In this case, their common boundary should be removed (see Figure 25). When variable $z_{t,p,q,r} = 1$ and $p = q$, we define the length of their common boundary to be 0 because we shall not remove any. Note that time $t \in T \setminus \{1, n\}$. We do not need $z_{t,p,q,r}$ for time $t = 1$ because there are no two polygons in the same patch. Namely, it always holds $z_{1,p,q,r} = 0$, which does not help in our ILP. We do not need $z_{t,p,q,r}$ for time $t = n$ because all the polygons will be in the same patch. In this case, Equation $z_{n,p,q,r} = 1$ always holds, which does not help in our ILP, either.

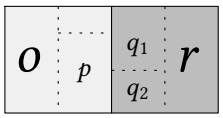


Fig. 26. There are two patches, which respectively use polygons o and r as their centers. Polygons in the same patch are separated by dotted lines. Polygon p, in patch o, has two neighbors assigned to center r, i.e., polygons q_1 and q_2. In this case, patches o and r are neighbors and can be aggregated.

We use a fourth set of variables to guarantee contiguity of each patch. In other words, we aggregate two patches only when they are neighbors (adjacent). We introduce

$$c_{t,p,o,r} \in \{0,1\} \qquad \forall t \in T \setminus \{n-1, n\}, \forall p, o, r \in P \text{ with } o \neq r,$$

with the intended meaning $c_{t,p,o,r} = 1$ if and only if, at time t, polygon p is assigned to center o, and p has a neighbor assigned to center r (see Figures 25 and 26 for examples). We do not need variable $c_{t,p,o,r}$ for time $t = n-1$ because there are only two patches left, and they must be neighbors.

Our last set of variables is needed to enforce that every aggregation step involves a smallest patch. We define

$$w_{t,o} \in \{0,1\} \qquad \forall t \in T \setminus \{n\}, \forall o \in P$$

with $w_{t,o} = 1$ meaning if and only if, at time t, patch o is the smallest patch that is involved in the aggregation step from time t to time $t-1$ (see for example Figure 25).

In total, the number of variables in our ILP formulation is $O(n^4)$.

A.2 Objective

We want to minimize a weighted sum of the two costs, the cost of type change and the cost of length (analogous to Equation 10). That is, our objective is to

$$\text{minimize} \quad (1-\lambda)F_{\text{type}} + \lambda F_{\text{lgth}},$$

where λ, as in Equation 10, is a parameter to assign importances of F_{type} and F_{lgth}. According to the cost introduced in Section 4.1, we compute the total cost of type change by

$$F_{\text{type}} = \sum_{t=2}^{n} \sum_{p \in P} \sum_{o \in P} \sum_{r \in P} \left(\frac{a_p}{A_R} \cdot \frac{d_{\text{type}}\left(T(o), T(r)\right)}{d_{\text{type\_max}}} \cdot y_{t,p,o,r} \right),$$

where, similar to Equation 1, a_p is the area of polygon p, A_R is the area of the region, and $T(o)$ and $T(r)$ are the types of centers (polygons) o and r.

We also wish to minimize the overall interior lengths of all the intermediate subdivisions. As discussed in Section 4.3, we use the length of the interior boundaries as an alternative to compactness. Recall that $B(P_{\text{start}})$ is the set of interior boundaries at time $t = 1$ (see Section 4.3). We sum up the normalized lengths of the remaining interior boundaries of all the intermediate subdivisions by

$$F_{\text{lgth}} = \frac{1}{n-2} \sum_{t=2}^{n-1} \frac{\sum_{b \in B(P_{\text{start}})} |b| - \frac{1}{2} \sum_{p \in P} \sum_{q \in P} \sum_{r \in P} \left(|b_{pq}| \cdot z_{t,p,q,r} \right)}{D(t)}, \tag{25}$$

where variable b_{pq} represents the common boundary between polygons p and q. We define the length of the common boundary to be 0 (i.e., $|b_{pq}| = 0$) if $p = q$ because there is no boundary to be removed in this case. Function $D(t)$, defined by Equation 7, is used to normalize the cost of length. As in Equation 6, we use denominator $n-2$ to balance between the cost of type change and the

cost of length. Integrating Equation 7 into Equation 25, we have

$$F_{\text{lgth}} = \frac{n-1}{n-2} \sum_{t=2}^{n-1} \left(\frac{1}{n-t} - \frac{\sum_{p \in P} \sum_{q \in P} \sum_{r \in P} \left(|b_{pq}| \cdot z_{t,p,q,r} \right)}{2(n-t) \sum_{b \in B(P_{\text{start}})} |b|} \right).$$

A.3 Constraints

In order to formulate our aggregation problem as an ILP, we restrict the variables introduced in Section A.1 by setting up constraints. Recall that the intended meaning of $x_{t,p,r} = 1$ is if and only if polygon p is assigned to center r at time t. To realize this functionality, our first constraint is that polygon p is assigned to exactly one center at time t. To this end, we require that

$$\sum_{r \in P} x_{t,p,r} = 1 \qquad \forall t \in T, \forall p \in P. \tag{26}$$

The next constraint is that polygon r is available to be assigned by other polygons only when r is a center. In our case, if polygon r is a center, then it must be assigned to itself, that is, $x_{t,r,r} = 1$. If r is not a center, we have variable $x_{t,r,r} = 0$. In either case, we have

$$x_{t,p,r} \leq x_{t,r,r} \qquad \forall t \in T, \forall p, r \in P. \tag{27}$$

Aggregating a patch into another one results in the number of centers decreasing by 1. We achieve that exactly one patch is aggregated into another by specifying the number of centers for each point in time, that is,

$$\sum_{r \in P} x_{t,r,r} = n - t + 1 \qquad \forall t \in T, \tag{28}$$

where polygon r is a center at time t if and only if $x_{t,r,r} = 1$.

When a patch is aggregated into another one, the center of the former will not be used as a center anymore. Hence, we have

$$x_{t,r,r} \leq x_{t-1,r,r} \qquad \forall t \in T \setminus \{1\}, \forall r \in P. \tag{29}$$

On the start map, there are some polygons with the goal type, T_{goal} (see definition in Section 7.1). At time $t = n$, all polygons are aggregated into one patch. This patch must have type T_{goal}. In other words, the center of this patch must be one of the polygons with type T_{goal} on the start map:

$$\sum_{r \in P:\ T(r) = T_{\text{goal}}} x_{n,r,r} = 1, \tag{30}$$

where $T(r)$ is the type of polygon r at time $t = 1$.

Next, we restrict binary variable $y_{t,p,o,r}$, introduced in Section A.1. Recall that the intended meaning of $y_{t,p,o,r} = 1$ is if and only if polygon p is assigned to center o at time $t-1$ and to center r at time t. To enforce this, we use two types of constraints.

First, if polygon p is assigned to center o at time $t-1$ ($x_{t-1,p,o} = 1$) and assigned to center r at time t ($x_{t,p,r} = 1$), we have variable $y_{t,p,o,r} = 1$. This requirement is expressed by

$$y_{t,p,o,r} \geq x_{t-1,p,o} + x_{t,p,r} - 1 \qquad \forall t \in T \setminus \{1\}, \forall p, o, r \in P. \tag{31}$$

Second, if p is not assigned to o at $t-1$ ($x_{t-1,p,o} = 0$) and/or p is not assigned to r at time t ($x_{t,p,o} = 0$), we have variable $y_{t,p,o,r} = 0$. This requirement is expressed by

$$\left.\begin{aligned} y_{t,p,o,r} &\leq x_{t-1,p,o} \\ y_{t,p,o,r} &\leq x_{t,p,r} \end{aligned}\right\} \qquad \forall t \in T \setminus \{1\}, \forall p, o, r \in P. \tag{32}$$

In Section A.1, we introduced binary variable $z_{t,p,q,r}$. Recall that the intended meaning of $z_{t,p,q,r} = 1$ is if and only if polygons p and q are both in patch r at time t. To enforce this, we need three types of constraints.

First, if two polygons p and q are assigned to center r at time t ($x_{t,p,r} = 1$ and $x_{t,q,r} = 1$), we have variable $z_{t,p,q,r} = 1$. This requirement is expressed by

$$z_{t,p,q,r} \geq x_{t,p,r} + x_{t,q,r} - 1 \qquad \forall t \in T \setminus \{1, n\}, \forall p, q, r \in P. \tag{33}$$

Second, at time t, if p is not assigned to r ($x_{t,p,r} = 0$) and/or q is not assigned to r ($x_{t,q,r} = 0$), we have variable $z_{t,p,q,r} = 0$. This requirement is expressed by

$$\left.\begin{array}{l} z_{t,p,q,r} \leq x_{t,p,r} \\ z_{t,p,q,r} \leq x_{t,q,r} \end{array}\right\} \qquad \forall t \in T \setminus \{1, n\}, \forall p, q, r \in P. \tag{34}$$

Third, we introduce an abbreviation that will be helpful to express the last type of constraint involving variable $z_{t,p,q,r}$:

$$z_{t,p,q} = \sum_{r \in P} z_{t,p,q,r} \qquad \forall t \in T \setminus \{1, n\}, \forall p, q \in P, \tag{35}$$

where the reason we do not need $z_{t,p,q}$ for $t = 1$ or $t = n$ is the same as for $z_{t,p,q,r}$ (see Section A.1). Variable $z_{t,p,q}$ expresses whether, at time t, polygons p and q are in the same patch ($z_{t,p,q} = 1$) or not ($z_{t,p,q} = 0$). Note that constraints (26) and (34) ensure that polygons p and q can be assigned to one common center at most; therefore, we have $z_{t,p,q} \leq 1$. We use our new variable $z_{t,p,q}$ to express the following requirement: If two polygons have been aggregated into one patch, they will always be in the same patch at later times—although the center of their common patch may change. In other words, variable $z_{t,p,q}$ is monotonically increasing as a function of time t:

$$z_{t,p,q} \geq z_{t-1,p,q} \qquad \forall t \in \{3, 4, \ldots, n-1\}, \forall p, q \in P. \tag{36}$$

Now we present our constraints of ensuring contiguity inside a patch. This problem has received considerable attention in integer linear programming. Usually, a subdivision is represented by a graph (see Figure 5). Zoltners and Sinha [83] regarded each node as a center. For each center, they found a shortest path to each of the other nodes. Then, they required that a center can be assigned by a node only if at least one immediate predecessor of the node in the shortest path had been assigned to the center. Although this requirement makes their problem easier to be solved, it excludes many feasible patches. Williams [79] built an optimal spanning tree for the nodes. In order to ensure contiguity, the method picks a user-specified number of nodes that constitute an optimal subtree of the previously built spanning tree. For a given center, Cova and Church [9] were able to find all the contiguous patches. In their method, when a node is to be assigned to a center, a path from the node to the center was demanded that each node of the path is assigned to the center. Similarly, Shirabe [60] modeled the contiguity problem as a network flow. He required that there must be a path so that some fluid can flow from a node to a sink (center). Oehrlein and Haunert [45] utilized a method based on *vertex separators*. Given center r and node p, a separator is a set of nodes such that any path from r to p will contain at least one node of the set. The contiguity between center r and node p is ensured if each of the separators contains at least one node assigned to the center. The last four ideas can be adapted into our method as we do not wish to exclude any possible solutions. However, we use an idea that is more intuitive for our problem since we aggregate step by step.

We aggregate two patches only if they are neighbors. To ensure this, we need binary variable $c_{t,p,o,r}$ introduced in Section A.1. Recall that the intended meaning of $c_{t,p,o,r} = 1$ is if and only if, at time t, polygon p of patch o has at least one neighboring polygon in patch r. To enforce this behavior of $c_{t,p,o,r}$, we need four types of constraints.

First, polygon p must actually be assigned to center o at time t ($x_{t,p,o} = 1$). In contrast, if p is not assigned to o ($x_{t,p,o} = 0$), then variable $c_{t,p,o,r}$ is impossible to tell if patch o and patch r are neighbors. In this case, we must not aggregate the two patches ($c_{t,p,o,r} = 0$); otherwise, we may end up having noncontiguous patches. As a result,

$$c_{t,p,o,r} \leq x_{t,p,o} \qquad \begin{array}{l} \forall t \in T \setminus \{n-1, n\}, \\ \forall p, o, r \in P \text{ with } o \neq r. \end{array} \tag{37}$$

Second, at time t, at least one of polygon p's neighbor(s), say, polygon q has to be assigned to center r ($x_{t,q,r} = 1$). If not, then variable $c_{t,p,o,r}$ is impossible to tell if patches o and r are neighbors. Analogous to the condition of constraint (37), we have

$$c_{t,p,o,r} \leq \sum_{q \in N_{\text{nbr}}(p)} x_{t,q,r} \qquad \begin{array}{l} \forall t \in T \setminus \{n-1, n\}, \\ \forall p, o, r \in P \text{ with } o \neq r, \end{array} \tag{38}$$

where $N_{\text{nbr}}(p)$ represents the set of polygons adjacent to p.

Third, if polygon p is in patch o ($x_{t,p,o} = 1$) and p has at least one neighbor, say, polygon q in patch r ($x_{t,q,r} = 1$), then we must enforce variable $c_{t,p,o,r} = 1$ (according to the definition of this variable). We have

$$c_{t,p,o,r} \geq x_{t,p,o} + x_{t,q,r} - 1 \qquad \begin{array}{l} \forall t \in T \setminus \{n-1, n\}, \\ \forall p, o, r \in P \text{ with } o \neq r, \forall q \in N_{\text{nbr}}(p). \end{array} \tag{39}$$

Fourth, if we aggregate patch o into patch r from time $t-1$ to time t, we have variable $y_{t,o,o,r} = 1$ (see the definition of this variable in Section A.1). In this case, we must make sure that the two patches are actually neighbors at time $t-1$. That is to say, at least one of patch o's polygons has at least one neighbor in patch r at time $t-1$. If not, we have $y_{t,o,o,r} = 0$. That is, it holds

$$y_{t,o,o,r} \leq \sum_{p \in P} c_{t-1,p,o,r} \qquad \forall t \in T \setminus \{1, n\}, \forall o, r \in P \text{ with } o \neq r. \tag{40}$$

If we do not require that each aggregation step must involve a smallest patch, then we only need constraints (26)–(40) and variables $x_{t,p,r}$, $y_{t,p,o,r}$, $z_{t,p,q,r}$, and $c_{t,p,o,r}$. If we insist on involving a smallest patch at each step, then we need more variables and more constraints.

A.3.1 Aggregation involving a smallest patch. In order to make sure that each of our aggregation steps involves a smallest patch, we need another type of variable, $w_{t,o}$. Recall that the intended meaning of $w_{t,o} = 1$ is if and only if polygon o is the center of a smallest patch at time t. We will use this to enforce that this patch is involved in the aggregation step from time t to $t+1$. At any time t, we pick exactly one smallest patch (there can be many) and aggregate it with one of its neighbors; we do not care whether or not the neighbor is a smallest one. Therefore, we have

$$\sum_{o \in P} w_{t,o} = 1 \qquad \forall t \in T \setminus \{n\}. \tag{41}$$

Assume that patch o is the smallest patch involved in the aggregation step from t to $t+1$ and that we are aggregating patch o and another patch, say, r. There can be two cases. We aggregate o into r or aggregate r into o. In the first case, we have variable $y_{t+1,o,o,r} = 1$, and, in the second case, we have $y_{t+1,r,r,o} = 1$. Either of the two cases implies that polygon o is indeed a center at time t, that is, $x_{t,o,o} = 1$. In order to enforce that the aggregation step involves patch o and another patch, we must make sure $y_{t+1,o,o,r} = 1$ or $y_{t+1,r,r,o} = 1$ when $w_{t,o} = 1$. Consequently, we use the constraint

$$w_{t,o} \leq \sum_{r \in P \setminus \{o\}} \left(y_{t+1,o,o,r} + y_{t+1,r,r,o} \right) \qquad \forall t \in T \setminus \{n\}, \forall o \in P. \tag{42}$$

Now we need to make sure that patch o with $w_{t,o} = 1$ is indeed a smallest patch at time t. We define variable $A_{t,r}$ as the area of patch r at time t. That is, we have

$$A_{t,r} = \sum_{p \in P} a_p \cdot x_{t,p,r},$$

where a_p is the area of polygon p and, as viewed by the ILP, is a constant. Area $A_{t,r}$ is positive if and only if polygon r is a center at time t ($x_{t,r,r} = 1$). We define constant M as a very large number to help us construct the corresponding constraints. It suffices to set M to the area of the whole region, i.e., $M = A_R$ (see Equation 1). We require

$$A_{t,o} - M(1 - w_{t,o}) \leq A_{t,r} + M(1 - x_{t,r,r}) \qquad \begin{array}{l} \forall t \in T \setminus \{n\}, \\ \forall o, r \in P \text{ with } o \neq r. \end{array} \tag{43}$$

This constraint takes effect only when $w_{t,o} = 1$ and $x_{t,r,r} = 1$, which indicates that patch o is smaller than or equal to all the other existing patches at time t.

In order to compute an aggregation sequence involving a smallest patch at each step, we need all the five types of variables and all the constraints (26)–(43). In total, the number of constraints is $O(n^4)$.

Received February 2007; revised March 2009; accepted June 2009