JOURNAL OF SPATIAL INFORMATION SCIENCE
Number 11 (2015), pp. 53–79

doi:10.5311/JOSIS.2015.11.230

RESEARCH ARTICLE

Routes visualization: Automated placement of multiple route symbols along a physical network infrastructure

Jules Teulade-Denantes[1,2], Adrien Maudet[1,3], and Cécile Duchêne[1,3]

[1]Laboratoire COGIT, IGN, Saint-Mandé, France
[2]Université Paul Sabatier, Toulouse, France
[3]Université Paris-Est, France

Received: April 15, 2015; returned: June 1, 2015; revised: August 18, 2015; accepted: September 17, 2015.

Abstract: This paper tackles the representation of routes carried by a physical network infrastructure on a map. In particular, the paper examines the case where each route is represented by a separate colored linear symbol offset from the physical network segments and from other routes—as on public transit maps with bus routes offset from roads. In this study, the objective is to automate the placement of such route symbols while maximizing their legibility, especially at junctions. The problem is modeled as a constraint optimization problem. Legibility criteria are identified and formalized as constraints to optimize, while focusing on the case of hiking routes in a physical network composed of roads and pedestrian paths. Two solving methods are tested, based on backtracking and simulated annealing meta-heuristics respectively. Encouraging results obtained on real data are presented and discussed.

Keywords: automated cartography, route maps, symbol placement, constraint formulation, constraint optimization problem, simulated annealing

1 Introduction

Maps can indicate specific paths, journeys, rides, or trails to follow. This is what we call routes. Routes can be represented in many ways on maps. The most common representation for hiking routes on topographic maps is a unique linear symbol, with identical specifications (color, width) for all routes. Labels enable the different routes to be distinguished. This representation is found, for example, on the maps produced at IGN (Institut National de l'Information Géographique et Forestière), the French national mapping agency, as shown on Figure 1 (left). This representation allows maps to be multipurpose without being overloaded. However it is not easy to distinguish each route independently in Figure 1 (left, for example, to trace the different routes "GR5," "GR533," "GR534"). A network of "road segments carrying routes" is perceived instead.

In this paper we are interested in another kind of route representation where each route is symbolized by a different symbol, distinguished by its color (or another graphical variable). The route symbols are positioned in relation to the other routes and to the carrying road. This representation occurs on some public transit maps where each bus line has its own symbol. It provides a global view of all the routes on the map, each route is perceived as a whole. Figure 1 (right) presents a map with such a representation. This representation has also been proposed recently in experimental maps designed for outdoor activities in the context of the MenoMaps project [15].

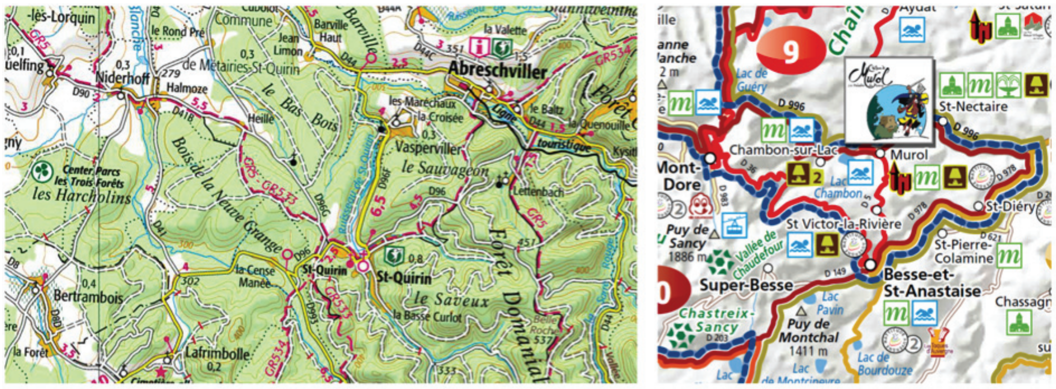

Figure 1: Left: IGN-France 1:100k topographic map. Right: Map showing touristic routes for motorbikes (CRDT Auvergne / Actual).

This second representation is less common on hiking maps, notably because no automated process exists to arrange the routes along the roads, so they have to be arranged manually. This paper studies how to automate the creation of maps using this second representation, starting from vector geographic data. First thoughts lead to the idea that the route positions along each carrying road have to be chosen to satisfy legibility constraints. Therefore we used the "constraint satisfaction problem" (CSP) paradigm to solve this problem. Note that other problems of the automated cartography domain have already been expressed as problems that seek to satisfy constraints, e.g., generalization as first proposed by [1], or label placement [5].

The work reported in this paper focuses on the case of hiking routes carried by a physical network infrastructure composed of roads and pedestrian paths. Although the constraints were designed with the hiking routes case in mind, the method could be applied for other types of routes or circuits carried by a physical network infrastructure (see also Section 6). This includes freight transport routes on airways, navigable waterways, road and/or railway networks; electric cables sharing physical conduits; more generally any type of route or trajectory followed by individuals, possibly pre-processed by aggregation into flows; etc.

We identify three main parts in our problem: the constraint identification, the problem modeling in the CSP paradigm, and the definition of solving methods. The rest of the paper is organized as follows. Related works are introduced in Section 2. The proposed approach to model the problem, including the constraint identification, is described in Section **??**. Two solving methods are described in Section 4. Section 5 presents an implementation and results on real data. Finally, Section 6 concludes and draws perspectives for further work.

2 Related works

2.1 Route representations

The route representation is governed by perceptual concepts. The Gestalt psychology introduced by Wertheimer [24] studies how the spatial arrangement of objects leads the human mind to perceive a subset of objects a whole. The Gestalt laws are recognized as interesting to improve maps legibility (see, e.g., [2]). In the automated cartography domain, they have especially been promoted for generalization [6]. In particular, the "good continuation" Gestalt principle has been successfully applied in automated road network generalization [12, 18, 19, 23]. In our case, we seek to represent routes so that each of them is perceived as a whole. Routes should respect the Gestalt laws of grouping, especially the laws of similarity (one color for one route), continuity (routes should stay as much as possible at the same position along their carrying road), and proximity (routes carried by a road should be drawn adjacent to each other and to the road). Minimizing the number of crossings is a known graph theory problem. Nöllenburg [14] references many methods to automate metro maps layout. In order to study them, he proposes to describe a set of design principles to create legible metro maps. All these principles apply to metro maps (or more generally to schematic maps), and none addresses the minimization of crossings independently from other problems, specific to metro maps. Among recent studies on this topic, Fink and Pupyrev [7] provide a complexity proof and approximate solving algorithms under certain working hypotheses. These working hypotheses are not valid in our case because only simple paths are considered whereas our problem contains closed paths. Therefore we cannot use anymore the extremity concept that constitutes the basis for the complexity proofs and algorithms.

2.2 Constraint optimization

The classic definition of a constraint satisfaction problem (CSP) is as follows [9]. A CSP is a triplet $\langle X, D, C\rangle$ where:

$X = \langle x_1, x_2, \ldots, x_n\rangle$ is a n-tuple of variables,
$D = \langle D_1, D_2, \ldots, D_n\rangle$ is a n-tuple of domains such that $x_i \in D_i$,
$C = \langle C_1, C_2, \ldots, C_t\rangle$ is a t-tuple of constraints.

A constraint C_j is expressed as a relation, defined on some subset of variables $X_{C_j} \subset X$, whose tuples are all the simultaneous value assignments to the members of this variable subset.

This formalism can for instance be used to formalize the benchmark problem of placing 8 queens on a chess board so that they do not threaten each other. The variables are the positions of the queens: ordered pairs of integers in the domain $[1..8] * [1..8]$. One constraint is that two queens cannot have the coordinates (i, j) and (i, k)—i.e., two queens cannot be placed on the same line.

A constraint optimization problem (COP) is a CSP with a score function that returns a value for each solution: the aim is not any more to completely satisfy all constraints, but to find an optimal solution. An optimal solution is a solution that minimizes (or maximizes) the score function. In our case, the variables should express the ordering (offsets) of itineraries along road segments and the constraints should express the legibility. As legibility is amongst others related to the number of crossings, and some crossings are not avoidable, we are in the optimization case. However, for COPs the difficulty lies in the creation of the score function [21].

Two classes of solving methods can be distinguished [20].

- The systematic algorithms offer the guarantee to find the optimal solution if it exists. They are mostly backtracking algorithms or dynamic programming. Backtracking algorithms explore all possibilities of variable assignations, in order to identify the best one.
- The non-systematic algorithms try to find an approximation of the optimal solution. They are used when the search space is too big for a systematic algorithm. Iterative stochastic methods are commonly chosen.

3 Problem statement

In this paper, we chose to work on a road network, composed of road segments (some of which can be footpaths in practice). A road segment corresponds to an edge of the network, it has a geometry that is a polyline and it is bordered by two roads junctions. In addition to this road network, we have a set of routes. For each route, we know the set of road segments that carry the route. A road segment can carry 0 to n different route segments (i.e., parts of routes carried by a same road segment). Figure 2a shows a road network composed of 5 road segments, and 2 routes (red and green) described by the road segments carrying them. Figure 2b gives a possible layout of the routes symbol. It can happen that a route is locally carried by no road segment, because the route is locally off-trail, or because the road segment that carries the route is missing in the road dataset. In this case, a fictive road segment is added to the road network. Hence any route segment is carried by one road segment.

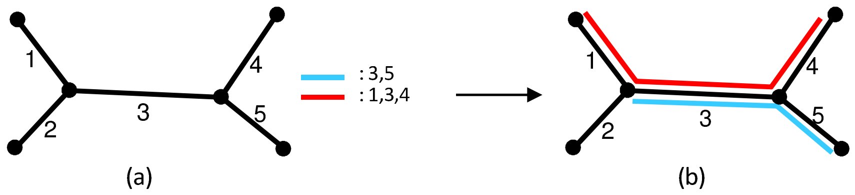

Figure 2: (a) A road network and 2 routes (b) a possible routes layout.

Moreover we made the following simplifications:

(1) Only roads and routes are considered; surrounding objects in the map (buildings, etc.) are ignored.
(2) The positions of route segments carried by a given road segment are not influenced by the presence of other roads approaching the road segment; they are only influenced by the configurations of the junctions that border this road segment (Figure 3).

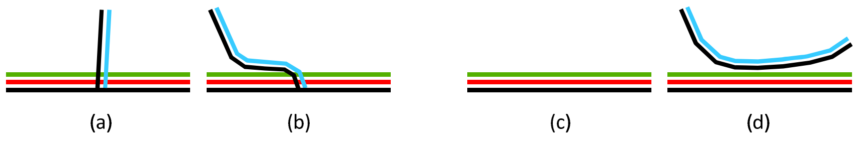

Figure 3: Problem simplification: configurations (a) and (b) are considered equivalent, configurations (c) and (d) are considered equivalent.

(3) The choice of colors for the route symbols is considered a separate problem and is not managed. For our tests, colors have been manually chosen after the route placement resolution.
(4) All routes are considered of equal importance, and displayed with the same symbol width. Therefore no constraint related to a route hierarchy or to symbol width aspects is defined.
(5) Once their positions have been computed, the route symbols are drawn with a simple method that, amongst others, does not manage which route symbol is drawn on top of which one at junctions (Figure 4). From interviews with cartographers working in production, we understood that when a route is crossing several other routes at a junction, it is better to draw this route on the top of the other ones (as on Figure 4a). We assume that it is possible to define a drawing method that respects that. Other characteristics of our drawing methods are described in Section 3.2.1.

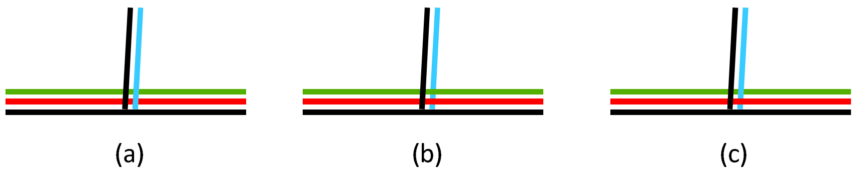

Figure 4: Problem simplification: for the shown route layout, any of the drawings (a), (b), or (c) can be obtained.

(6) The angles between road segments at junctions are not taken into account when evaluating the legibility of route positions, e.g., in Figure 5.

3.1 Considered constraints

In order to design a relevant set of the legibility constraints, we rely on the Gestalt laws. We identify the four following constraints:

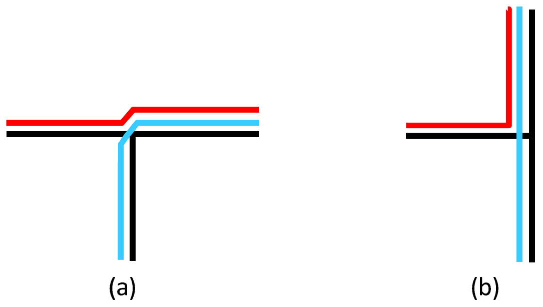

Figure 5: Problem simplification: situations (a) and (b) are considered equivalent.

- C1: All route segments carried by the same road segment have to be contiguous to each other and to the road segment. In other words, it is not allowed to have a gap between two route segments carried by a road segment, or between a route segment and the road segment (cf. the Gestalt proximity law). For example, on Figure 6 situation (a) is permitted, not situation (b).
- C2: The continuity on a same route (cf. the Gestalt continuity law) is ensured as much as possible. For example, in Figure 6d, all routes except the green one are continuous, while none is continuous in Figure 6c. Note that this constraint indirectly manages the Gestalt closure principles for routes forming loops, since only the presence of crossing leads to closure being unsatisfied. Decreasing the number of crossings implies improving the satisfaction of closure.
- C3: The number of crossings between the routes and between the routes and the roads is minimized. To illustrate the notion of crossing, Figure 6c counts 3 crossings, and Figure 6d counts 4 crossings.
- C4: The number of crossing clusters is minimized. A crossing cluster is a set of crossings that are not disjoint in terms of the involved routes, as defined by Fink and Pupyrev [8]. Figure 6c counts 2 clusters, while Figure 6d only counts 1 cluster, all surrounded by ellipses.

3.2 Problem variables modeling

A road segment carries route segments. The position of a route segment along a road segment is defined by an integer that can be interpreted depending on the road geometry: positive on the left of the road, negative on its right, increasing absolute value as the distance to road increases. Figure 7 shows two different configurations for the same road segment. The road segment is always situated at position 0. Our objective is to assign the most appropriate position for each route segment in order to satisfy the legibility constraints at best.

3.2.1 Handling route drawing at junctions

We assume that we have a drawing method that creates the suitable symbols (a strip with a defined color and a defined width) from the initial geometries of the roads and the positions

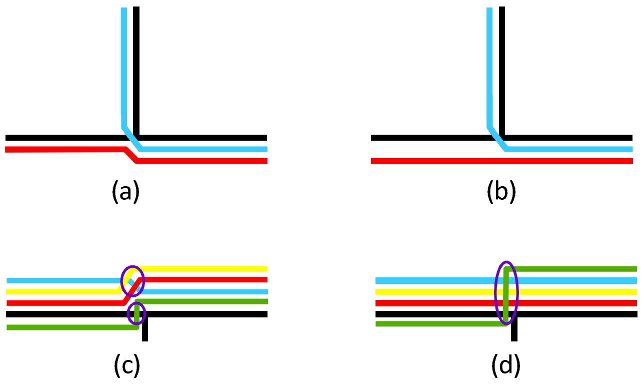

Figure 6: Illustration of considered constraints. Top: situation (b) not permitted (C1). Bottom: two route layouts where ellipses represent crossing clusters (see text for explanation of C2 to C4).

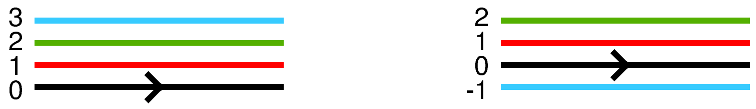

Figure 7: Two different route layouts for a road segment directed from left to right and carrying three routes. Route positions are defined by integer.

of the segments of the route along the roads, and that this method minimizes the crossings number at junctions for a given layout of the routes (on Figure 8, this drawing method would produce drawing (b) rather than (c)).

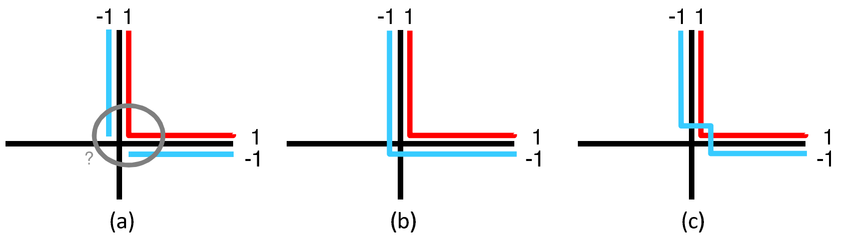

Figure 8: We assume we have a display method that minimizes the numbers of crossings at junctions for a given layout of the routes. For the layout (a), the produced drawing would be (b) rather than (c).

In practice, we set up a method to count the minimum number of crossings at a junction for a given layout (described in Section 3.3.2), but we did not write the drawing method and used a simpler one instead, which draws each route segment using a morphologic dilation of the road segment, and simply joins with a straight line the resulting route segments at junctions. Therefore, even when the route segments are optimally placed with respect to the number of crossings at a junction, some junctions are badly drawn in the map showing the result of the placement, as shown by Figure 9. This should be ignored when visually assessing the route symbols placement results.

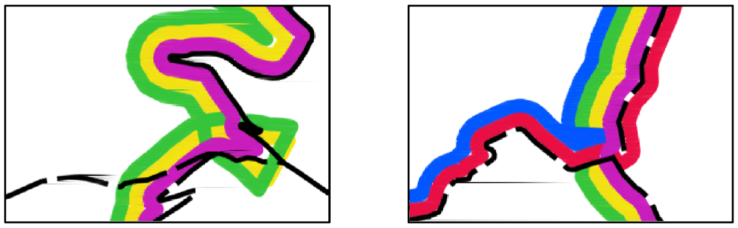

Figure 9: Here the route symbols placement is good, but the junctions are badly drawn because our drawing method is simpler than the optimal one.

3.2.2 Variables for the constraint optimization problem

To model our problem as a constraint optimization problem (COP), we define a variable for each road segment. This variable describes the layout of the route segments carried by the road segment. Thus, for a road segment carrying n route segments, the domain of the variable is the set of possible n-tuples corresponding to the possible layouts of the route segments carried by the road segment.

To decrease the size of the problem, and to ensure the satisfaction of constraint C1 (cf. Section 3.1), it was chosen to define domains for the variables that automatically satisfy C1, i.e., only n-tuples of contiguous positions are part of the domains. In other terms, C1 is necessarily satisfied by the domain definition. For example, for a road segment carrying 2 route segments, we have the following domain: $\{(-2,-1); (-1,-2); (-1,1); (1,-1); (1,2); (2,1)\}$. It is a set of 6 ordered pair, each ordered pair being a possible value for the 2 route segments carried by the road segment. The size of the set is equal to $(n+1)!$ for a road segment carrying n route segments.

We note D_n the domain of the variable corresponding to a road segment that carries n routes (e.g., D_2 is the domain composed of 6 ordered pairs shown in the above paragraph).

3.2.3 Network modeling refinement: Grouping road segments into "strokes"

With the road network modeling introduced in Section 3, we could theoretically end up with routes changing their position at every junction along a road, as illustrated in Figure 10a, while we would prefer to avoid it for legibility reasons. This is because only road segments are initially modeled, but higher order entities (usually called "roads" in the everyday language) are perceived due to continuity criteria as explained by the Gestalt theory. In the context of generalization, these "perceived roads" are called "strokes" by Thomson and Richardson [18], who first proposed to automatically build them from road segments, and to reason on them rather than on road segments for road network selection.

We reused this concept in the modeling of our problem: to avoid repetitive position changes along a perceived road, while reducing the problem complexity, adjacent road segments are gradually grouped into so-called "route-based road strokes" based on three criteria used in conjunction:

- the deflection angle at the junctions where the road segments meet is below a given threshold (in our case $102.5°$); this is a classical criterion to build a "stroke" as defined by Thomson and Richardson [18];
- the road segments have the same cartographic symbol; and
- the road segments carry route segments that belong to the same set of routes.

Figure 10 (b, c, d) shows examples of strokes built according to this logic. The computed "route-based road strokes" are used instead of the initial road segments in the constraint satisfaction problem modeling. This partially remedies to the simplification no. 5 described in Section 3 (for junctions of the road graph that are internal to a "route-based road stroke"). In the rest of the paper, initial "road segments" can be forgotten, and the term "road" will be used to refer to a "route-based road stroke." Variables as defined in previous section are associated to these roads. The term "route" will be used to refer to the part of a route carried by a road (called "route stroke" in the UML diagram below), unless otherwise stated.

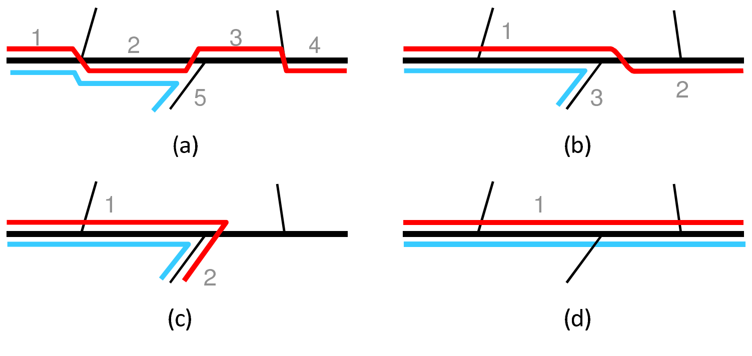

Figure 10: (a) Initial modeling: 5 road segments carrying routes; (b, c d) Road segments grouped into "strokes" on criteria of angles at junctions and carried routes: respectively 3, 2, and 1 road-route strokes for various locations of the routes.

Figure 11 shows the initial data model (road segments, routes and route segments) enriched with strokes: a road segment (*RoadSegment* class) carries 0 to n route segments (*RouteSegment* class), each associated with a position as described in Section 3.2; a route (*Route* class) is composed of route segments; a road stroke (*RoadStroke* class) is a group of oriented roads segments (*OrientedRoadSegment* class), which are road segments (inheritance relation) carrying a flag boolean (*orientationFlag* property) that enables the direction of the road segment to be compared to the direction of the stroke (same or reverse); a route stroke (*RouteStroke* class) is a group of route segments carried by a road stroke, it has a position on the road stroke (as described in Section 3.2, *positionOnRoadStroke* property); from this position, the position of its route segments on their road segment can be deduced (*getPositionOnRoadSegment()* method). During the resolution of the Constraint Optimization Problem, and during the rest of the paper, only the upper part of the class diagram (road strokes, also called "roads," carrying route strokes, usually simply called "routes") is used.

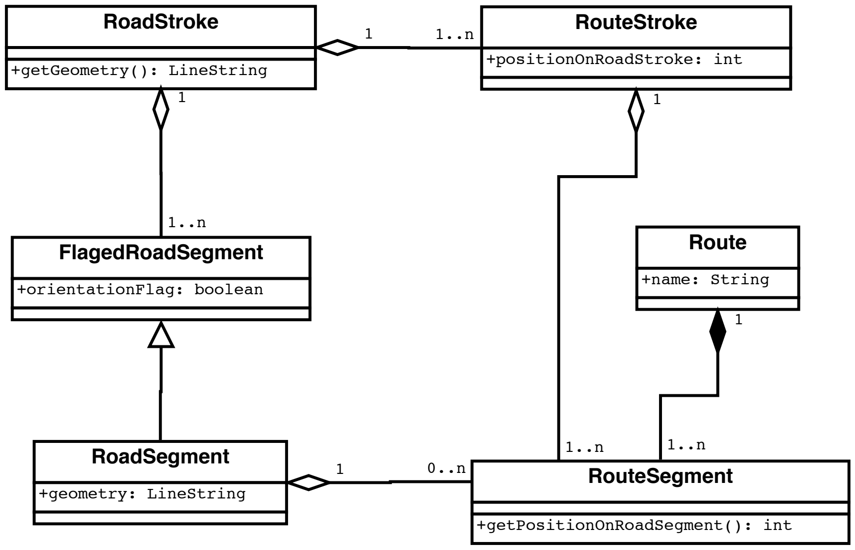
RoadStroke
+getGeometry(): LineString
RouteStroke
+positionOnRoadStroke: int
FlagedRoadSegment
+orientationFlag: boolean
Route
+name: String
RoadSegment
+geometry: LineString
RouteSegment
+getPositionOnRoadSegment(): int
1
1..n
0..n

Figure 11: Class diagram showing relations between road, routes, and strokes.

3.3 Constraint evaluation

In this section, we explain how we evaluated the considered legibility constraints in the context of a minimization problem. Constraint C1 is managed by the domain definition as explained previously; therefore we have to evaluate constraints C2, C3, and C4. These constraints have been described at an abstract level in Section 3.1. They can be considered constraint types that are then instantiated into constraint instances in the map. The term "constraint" hereafter refers to those constraint instances. As will be explained, some of these constraints can be evaluated at the level of a road, others have to be evaluated at the level of a junction, i.e., they concern all the roads that meet at that junction. A constraint of the latter kind can only be evaluated once the variables associated to all concerned roads have been assigned.

For each constraint type, a score function that associates a cost to each constraint instance is defined, representing the constraint violation: the highest the score or cost is, the less satisfied the constraint is. Solving (optimizing) the constraint optimization problem aims at reaching a minimum cost for the constraints. More precisely we seek to minimize a global cost function defined as the sum of the costs computed for all constraints (cf. Section 3.4). The cost function for each constraint type was defined theoretically while analyzing a few "toy situations" (i.e., typical situations defined after studying some maps). In order to define the balance between the costs of the different constraints, we always kept in mind, while defining each cost function, that the costs would eventually be aggregated by means of a sum. Constraints and their cost functions were then tested and tuned, separately and relatively to each other, using a few typical situations extracted from our real data set. The balance between costs of different constraints is discussed more in next subsections and in Section 3.4.

The descriptions of the constraints will use the following notations:

- x_{rd} is the variable associated to road *rd*. Using the definition of D_n given in Section 3.2.2, the domain associated to variable x_{rd} is $D_{n_{rd}}$, where n_{rd} is the number of routes carried by road *rd*.
- $\bar{x}_{rd} = (\bar{x}^1_{rd}, \ldots, \bar{x}^{n_{rd}}_{rd})$ is used to denote the value assigned to variable x_{rd}. $\bar{x}_{rd} \in D_{n_{rd}}$.
- X is the tuple of all variables to assign. It contains the variables corresponding to all roads of the considered map that carry routes: $X = (x_{rd})_{rd \text{ carrying routes}}$. The domain associated to X is $\Pi_{(rd \text{ carrying routes})} D_{n_{rd}}$. An assignment of this tuple is noted $\overline{X}$; $\overline{X} \in \Pi_{(rd \text{ carrying routes})} D_{n_{rd}}$.
- For a junction j, X_j is the tuple of the variables associated to all roads adjacent to j: $X_j = (x_{rd})_{rd \text{ adjacent to } j}$. An assignment of X_j is noted $\overline{X}_j$; $\overline{X}_j \in \Pi_{(rd \text{ adjacent to } j)} D_{n_{rd}}$.
- A constraint c is defined as a pair (X_c, f_c), where:

  X_c is the tuple of the variables associated to the roads concerned with the constraint; $\overline{X}_c \in \Pi_{(rd \text{ concerned by } c)} D_{n_{rd}}$.

  $f_c\colon \Pi_{(rd \text{ concerned by } c)} D_{n_{rd}} \to \mathbb{R}$ is the score function associated to c, it associates a real score value to any assignment $\overline{X}_c$ of X_c. This function is built as a cost function: the lower the score, the better the constraint is satisfied.

To simplify the understanding of the link between the constraints described in Section 3.1, and our notations, we kept the number of the constraint as subscript of the defined cost functions and the constants they use.

3.3.1 Constraint C2—Fostering route continuity

Constraint C2 concerning route continuity is evaluated at the level of a junction (one constraint at each junction where routes are present). For each route passing through this junction, the distance between the positions of the route on each side of the junction is evaluated. In Figure 12c, the green and the blue routes have a distance of 0, the red route a distance of $|3 - (-1)| = 4$.

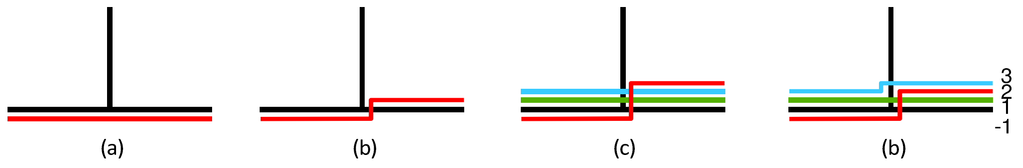

Figure 12: The red route changes respectively from position -1 to position -1 (a), 1 (b), 3 (c), and 2 (d).

The particular case where a route is branching at a junction (for routes that are not simple paths) is handled as follows: for each segment of this route that meet at this junction, a partial distance is computed as the minimum difference of position with other segments of the route; the aggregated distance for the junction is then the sum of the partial distances. This relies on the idea that if at least two segments of the route are at the same position, a kind of continuity is ensured and the problem is less serious than when a shift exists between two segments of a regular (non-branching) route.

To formalize this, we define the function *minimum_distance*(rt, j, X_j), which gives the minimum distance between two segments of the route *rt* at junction j, knowing the assignment $\overline{X}_j$ of the tuple of variables X_j, (i.e., assignment of the routes on all roads adjacent to j).

To evaluate the score of the constraint for a given route, a logarithmic transformation is applied to the computed distance. [1] This slows the increase of the score as distance increases. For instance, for the red route the score difference will be greater between Figure 12a (distance 0) and Figure 12b (distance of 2) than between Figure 12b (distance of 2) and Figure 12c (distance of 4). In other terms, we assume that, if we allow a shift (discontinuity) for a route, it is not much more critical to allow a bigger shift. The total score of the constraint for the junction is the sum of the scores obtained for each route passing at the junction. This way, the score of the constraint for the junction will be greater (worse) for situation Figure 12d than for situation Figure 12c (one additional discontinuous route vs one route that is more discontinuous).

In terms of formalization, a constraint c_{2_j} is defined for each junction j as follows:

$$c_{2_j} = (X_j, f_{2_j} : \Pi_{(\textit{rd adjacent to } j)} D_{n_{rd}} \to \mathbb{R}),$$

where the score function f_{2_j} is computed as

$$f_{2_j}(X_j) = \lambda_2 * \sum_{\textit{rt passing through } j} \ln\big(\textit{minimum\_distance}(rt, j, X_j) + 1\big),$$

where $\ln$ represents the natural logarithm (base e), and $\lambda_2 \in \mathbb{R}^+$.

Coefficient λ_2 is a weight that enables the cost of the constraint to be tuned with respect to other constraints. The function has the value of 0 for a junction where all routes are continuous. Each discontinuous route at the junction adds a penalty of $0.7 * \lambda_2$ ($= \lambda_2 * \ln(2)$) for a discontinuity (minimum distance) of 1, a penalty of $1.1 * \lambda_2$ ($= \lambda_2 * \ln(3)$) for a discontinuity of 2, and so on with a logarithmic increasing. In our implementation the value of λ_2 has been fixed to 2, which can only be interpreted compared to weights fixed for other constraints (see following subsections).

3.3.2 Constraint C3—Minimizing crossings

Constraint C3 tends to minimize the crossings between a route, other routes, and roads. We distinguish two kinds of crossings: internal crossings along the road carrying the route, and crossings at roads junctions. Therefore we refine constraint C3 into two more specific constraints, respectively called C3a (for internal crossings) and C3b (for crossings at junctions).

Constraint C3a on crossings occurring within a road (recall that "road" stands for "route-based road stroke") is evaluated at the road level—one constraint per road carrying routes. Only route/road crossings are concerned, as route/route crossings within a road are already taken into account by constraint C2. Possible situations can be reduced to two configurations: routes are present on one side of the road, or both sides. We assume that the best route layout is the situation where all routes are on the side of the road where they would cross the fewer roads (in Figure 13, situation (d) with 2 crossings). The score of constraint C3 is computed as the difference between the numbers of crossed roads (on both

[1] We arbitrarily chose to use the natural logarithm (base e). Another logarithm could be used but the weighting of the cost function, i.e., the λ_2 coefficient introduced below, should be adapted accordingly.

sides of the carrying road) for the current layout and for the best layout. Note that a crossing with a given road on a given side of the carrying road is only counted once regardless the number of routes that are present on this side, since all routes will be interrupted in the same way and the fact that more routes are concerned does not make it more difficult to mentally reconstruct them. As an example, in Figure 13 situations (a) and (b) are evaluated the same with a score of 1 ($= |3-2|$) for C3a, while in situation (c) C3a has a score of 3 ($= |5-2|$).

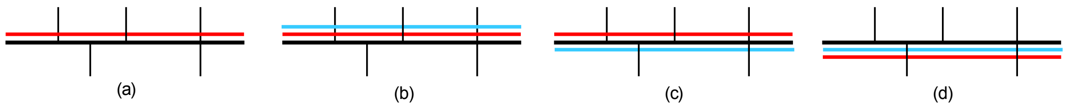

Figure 13: Evaluation of crossings along a road: configurations (a) and (b) are considered equivalent while (c) is considered worse because more roads are crossed; (d) is the best possible configuration.

The formalization of constraint C3a for a road *rd* is as follows:

$$c_{3a_{rd}} = ((x_{rd}), f_{3a_{rd}} \colon D_{n_{rd}} \to \mathbb{R}),$$

where the value of the score function $f_{3a_{rd}}$ is computed as

$$f_{3a_{rd}}(x_{rd}) = \begin{cases} \lambda_{3a} * \left(\#_{\text{left}}(rd) - \min(\#_{\text{left}}(rd), \#_{\text{right}}(rd))\right) & \text{if } \forall i \in [1, n_{rd}],\ \bar{x}^i_{rd} > 0, \\ \lambda_{3a} * \left(\#_{\text{right}}(rd) - \min(\#_{\text{left}}(rd), \#_{\text{right}}(rd))\right) & \text{if } \forall i \in [1, n_{rd}],\ \bar{x}^i_{rd} < 0, \\ \lambda_{3a} * \left(\#_{\text{left}}(rd) + \#_{\text{right}}(rd) - \min(\#_{\text{left}}(rd), \#_{\text{right}}(rd))\right) & \text{otherwise}, \end{cases}$$

with $\#_{\text{left}}(rd)$ (resp. $\#_{\text{right}}(rd)$) the number of roads crossed on the left side (resp. the right side) of *rd*; and $\lambda_{3a} \in \mathbb{R}^+$.

Coefficient λ_{3a} is a weight that enables the cost of the constraint to be tuned with respect to other constraints. The computed cost has a value of 0 for situations where all routes are carried on the side of the road with the less roads crossed. Each additional crossed road compared to this situation adds a penalty of λ_{3a}. In our implementation, λ_{3a} has been fixed to 0.5, which, compared to the value of λ_2 (for constraint C2), means that crossing up to two additional roads (penalty of 1) is preferred to adding one discontinuity of distance 1 (penalty of 1.4).

Constraint C3b on crossings occurring at a junction is evaluated at the level of a junction (one constraint at each junction where routes are present). In the same way as for constraint C3a, the score for constraint C3b for a given layout of the routes is computed as the difference between the number of route/route and route/road crossings for this layout, and the number of route/route and route/road crossings for the layout(s) that minimize this number of crossings (considered the best layout for C3b). The number of crossings for the best layout(s) is computed once for each junction before the optimization process starts, while exploring all possible layouts for the junction. As only the roads involved in the junction are concerned, the number of combinations to explore remains reasonable.

To count the number of crossings, we were inspired by the vertex crossings model from Fink and Pupyrev [7]. The mechanism is described in Figure 14a. For this example, we use the abstraction shown on Figure 14b, which only considers the order in which roads and routes appear as we turn around the junction. With this abstraction, we get the cycle:

[Black, Blue, Yellow, Green, Purple, Red, Blue, Red, Purple, Black, Green, Purple, Yellow, Black] where Black is the road, here handled as if it was a route. Note that this list is actually a cycle; here we arbitrarily listed the sequence clockwise starting from the road on the left in Figure 14a (black edge on the bottom-right in Figure 14b). Crossings will be counted iteratively for each route (here "route" stands for a route in its whole, not for a segment). Every time the crossings of a route have been counted, the route is removed from the cycle. A sequence to process the routes is randomly chosen. Here: Green $\rightarrow$ Purple $\rightarrow$ Yellow $\rightarrow$ Red $\rightarrow$ Blue.

For the green route, the cycle described above is split with respect to the green route, which gives us two sets (Figure 14c):

$$S_1 = \{\text{Purple, Red, Blue, Black}\} \quad \text{and} \quad S_2 = \{\text{Black, Blue, Yellow, Purple}\}$$

The routes crossing the green route are the routes that are present in both sets (this can easily be understood from Figure 14c). In our example, we count 3 crossings for the green route (with the Purple, Black, and Blue routes).

After processing the green route, we remove it from the current cycle. The next route being processed is the purple one. As it is branching at this junction, it appears three times in the cycle and gives us the following three sets (Figure 14d):

$$S_1 = \{\text{Red, Blue}\}, \quad S_2 = \{\text{Black}\} \quad \text{and} \quad S_3 = \{\text{Black, Blue, Yellow}\}.$$

The purple route has one crossing with a given route if this route appears in two out of the three sets. If a route appeared in the three sets (which can occur if the route is also branching at the junction), there would be two crossings with this route. For a given route A, counting the crossings with a route B can be generalized as follows: splitting the cycle with respect to A gives us n sets corresponding to sectors delimited by route A at the junction, which can be ordered cyclically around the junction; the sets containing at least one occurrence of route B are marked; the number of crossings between routes A and B is the length of the shortest path joining all marked sets within the cycle formed by the sets. In our case, the number of sets never exceeds 4 (route that makes a "loop").

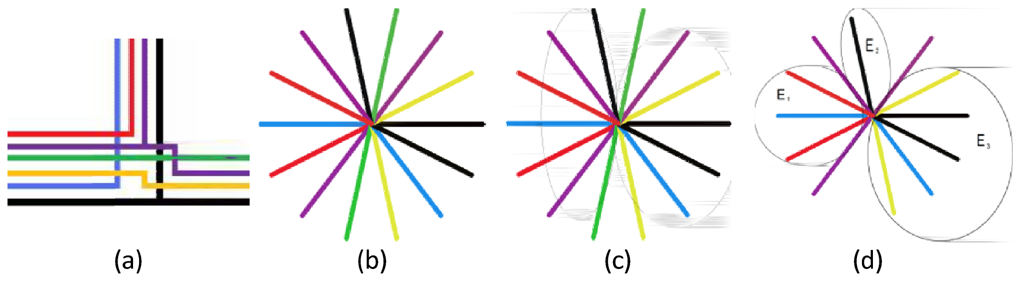

Figure 14: Example of crossings counting for junction (a). We get the corresponding abstraction (b), and the ellipses for the green route (c) and for the purple route (d).

The counting process is iterated until all routes have been processed. In our example, we count 8 crossings $(3 + 2 + 2 + 2 + 1 + 0)$.

To formalize this, we define the function *count_crossings*(j, X_j), which gives this value for a junction j and a given assignment $\overline{X}_j$ of the tuple of variables X_j.

For each junction j, we define a constraint c_{3b_j} as follows:

$$c_{3b_j} = (X_j, f_{3b_j} : \Pi_{(rd\ adjacent\ to\ j)} D_{n_{rd}} \rightarrow \mathbb{R}),$$

where the score function f_{3b_j} is computed as

$$f_{3b_j}(X_j) = \lambda_{3b} * count\_crossings(j, X_j)$$

with $\lambda_{3b} \in \mathbb{R}^+$.

Coefficient λ_{3b} is a weight that enables the cost of the constraint to be tuned with respect to other constraints. The computed cost has a value of 0 for situations where no crossing occurs at the junction. Each crossing adds a penalty of λ_{3b}. In our implementation, λ_{3b} has been fixed to 0.6. Compared to the score function of constraint C3a (with a value of λ_{3a} fixed to 0.5), it means that we prefer to add one route/road crossing within a road rather than one route/road or route/route crossing at a junction. A comparison to the score function of constraint C2 (with a value of λ_2 fixed to 0.5) can be interpreted while looking at Figure 12 (c and d), page 63. Indeed, we prefer situation (c) to situation (d): we prefer to see one route (the red one) cross one more route (penalty $+0.5$) and increase its existing discontinuity from 3 to 4 (additional penalty $2*\ln(5)-2*\ln(4) = +0.4$), rather than to see a discontinuity appear on one more route (the blue one) (penalty $2 * \ln(2) = +1.4$).

3.3.3 Constraint C4—Minimizing the number of crossing clusters at junctions

Constraint C4 concerning the crossing clusters is evaluated at the level of a junction (one constraint at each junction where routes are present). The number of crossings clusters at a given junction is computed as follows: dynamically find the route with the most crossings, remove it and reiterate the process. When no more crossing can be found at the junction, the counting process terminates and the number of crossings clusters is equal to the number of removed routes.

A function *count_crossings_clusters*(j, X_j) is defined, which gives this value for a junction j and a given assignment of the tuple of variables X_j.

Moreover, we chose to penalize the presence of several crossing cluster on a same intersection, which is a factor of unreadability on an intersection. For this we used an exponential function to avoid a clusters accumulation.[2]

For each junction j, we define a constraint c_{4_j} as follows:

$$c_{4_j} = (X_j, f_{4_j} : \Pi_{(rd\ adjacent\ to\ j)} D_{n_{rd}} \rightarrow \mathbb{R}),$$

where the score function f_{4_j} is computed as

$$f_{4_j}(X_j) = e^{\lambda_4 * count\_crossing\_clusters(j, X_j)} - 1$$

with $\lambda_4 \in \mathbb{Z}^+$.

Contrary to the score functions of other constraints, we have not weighted the score function using a multiplication factor. Coefficient λ_4, set within the exponential function, enables the computed cost to increase more rapidly with the number of crossing clusters.

[2]We arbitrarily chose to use the natural exponential function (base e). Another exponentiation function could be used, but the weighting of the cost function, i.e., the λ_4 coefficient introduced below, should be adapted accordingly.

The computed cost has a value of 0 for situations where no crossing occurs at the junction. One crossing cluster gives a score of $2.7^{\lambda_4} - 1$ $(= e^{\lambda_4} - 1)$, two crossing clusters a score of $7.4^{\lambda_4} - 1$ $(= e^{2*\lambda_4} - 1)$, etc. In our case λ_4 has been set to 2, in order to hugely penalize (close to a veto) the presence of several crossing clusters, assessed as extremely bad for legibility, at a junction. Indeed, increasing the number of crossing clusters at a junction from 1 to 2 increases the cost computed for the constraint by 47.2 $(= (e^{2*2} - 1) - (e^{2*1} - 1))$, which, given the tuning of the score functions for other constraints, is equivalent to, for example, adding 78 intersections at a junction.

3.4 Aggregated cost function from defined constraints

The constraints defined in the previous section are used to evaluate the overall quality of a possible solution (complete assignment of variables, i.e., route positions along roads). More precisely, we do not seek to assess the quality of a solution on its own on an absolute scale, but to assess whether a solution is better or worse than another one. For this, we use a cost function that aggregates the evaluation of all instances of constraints. This function, called *Score()*, is computed as the sum of the scores computed for all instances of constraints. In other words, for an assignment of the variables X, the total score is computed as follows:

$$\mathit{Score}(X) = \sum_{\mathit{rd\ carrying\ routes}} f_{3a_{rd}}(x_{rd}) + \sum_{j \in \mathit{AllJunctionsWithRoutes}} \big(f_{2_j}(X_j) + f_{3b_j}(X_j) + f_{4_j}(X_j) \big)$$

The shapes of the different score functions are shown in Figure 15, on which visual comparison is enabled by a common scale of the scores (y axis) for C2, C3a, and C3b. The tuning of the functions with respect to each other has been discussed in Sections 3.3.1–3.3.3.

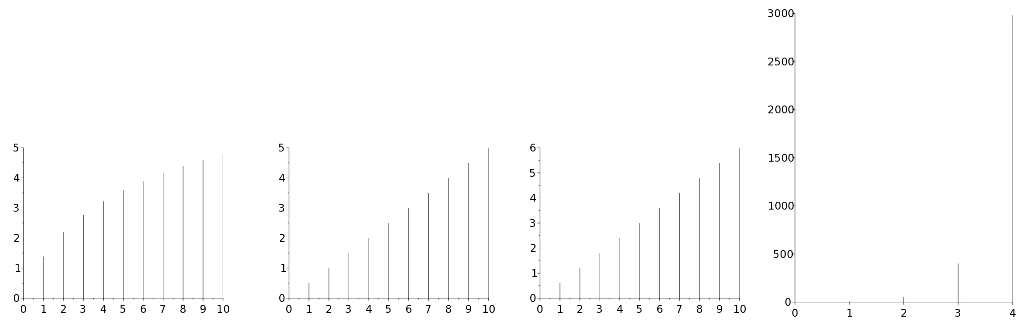

Figure 15: Shapes of the score functions defined for constraints C2 to C4. (a) shows $y = \lambda_2 * \ln(x+1)$, for C2; (b) shows $y = \lambda_{3a} * x$, i.e., the impact of crossing x additional roads, for C3a ; (c) shows $y = \lambda_{3b} * x$, for C3b; (d) shows $y = e^{\lambda_4 * x} - 1$, for C4.

4 Problem solution

4.1 Systematic approach

To optimize the identified constraints, we first used a backtracking algorithm. It is a depth-first search that explores as far as possible along each branch of the search tree before backtracking [20]. This kind of algorithm is relevant for an optimization problem if partial solutions can be assessed. Thus, subtrees containing no satisfying solutions can be pruned before being explored. As we are in the case of a constraint optimization problem, to find the optimal solution we used a *score* variable representing the function to optimize, i.e., the cost of the solution, as suggested by Smith [17]. In our case, the score to optimize is given by the *Score()* function defined in Section 3.4. When a solution is first reached, i.e., all variables have been instantiated, the *score* variable is instantiated with the score of this solution and a new constraint is added, stating that the value of the *score* variable must be better than the current value of *score* (in a minimization problem, better means lower). The value of *score* at the end of the exploration corresponds to the best reached score. Please note that at any node of the tree, a constraint associated to a junction can only be assessed if all variables associated to the junction (i.e., to the roads incident to the junction) have been assigned. Otherwise, the score of the partial solution associated to the node does not take the constraint into account. The corresponding pseudo code is presented in Algorithm 1.

Algorithm 1 Pseudocode for backtracking approach to optimization

```
$score \leftarrow initialScore$
$i \leftarrow 0$
while ($i > -1$)
    if ($allVariablesInstanciated(variables)$) then
        if ($currentScore() < score$) then
            $score \leftarrow currentScore()$
        $variables[i].resetValue()$
        $i--$
    else if ($currentScore() < score$) then
        $i++$
        $variables[i].instanciateValue()$
    else if ($variables[i].hasNext()$) then
        $variables[i].nextValue()$
    else
        $variables[i].resetValue()$
        $i--$
return ($variables, score$)
```

Next, we optimized the algorithm by initializing the *score* variable with an upper bound of the optimal score, in order to avoid a costly exploration at the beginning. This upper bound is reached while performing a random exploration of the research tree and keeping the lowest reached score. We also ordered the variables to prune the search tree earlier. For that, we used the notion of complexity of a road junction. A junction is more complex the larger number of possible layouts is can be associated with—the layout of the junction being the combination of the layouts of the roads incident to the junction. The number of possible layouts at a junction, i.e., the number of possible variables combinations associated to the junction (one road corresponds to one variable), can easily be computed from the number of roads incident to the junction and the numbers of routes carried by these roads. The idea is that the more complex junctions are associated to constraints that generate higher partial scores and therefore are more likely to enable to prune the search tree. Therefore we ordered the variables so that the variables associated to more complex junctions are instantiated first. Besides, note that at a given stage of the process, constraints taken into account are those corresponding to roads that have already been assigned, and those corresponding to junctions of which all adjacent roads have been assigned. Other constraints are not considered (their contribution is 0). There is no risk of pruning the tree too much and missing better solutions, as taking into account more constraints can only increase the total score.

Despite this, performance remained poor for our problem (cf. Section 5). However, this algorithm did allow our constraint system to be improved and validated by testing it on a restricted subset of variables corresponding to typical situations on a route map. Constraint C4 on crossing clusters, which had initially not been identified as a constraint, was added at this stage.

4.2 Non-systematic approach

As an alternative to the systematic approach, with performance that was not sufficient, we modeled a non-systematic approach to get better performance. We used the simulated annealing meta-heuristic [11], which is part of the iterative stochastic methods that are classically used to solve constraint optimization problems that have a large search space [20]. In the domain of automated cartography, simulated annealing has been used for label placement [5], generalization of urban areas at topographic scales [22], and color assignment to maximize contrasts on schematized flight route maps [10]. Simulated annealing seeks to minimize an *energy* computed from the variables to assign (the energy represents the criterion to optimize). It starts from a given initial solution (a given assignment of the variables). At any stage of the process, from the current solution a new solution is computed by applying a *newAssignment()* function that modifies part of the current solution. If this new solution enhances (i.e., decreases) the energy, it is accepted as the new current solution. If the energy is increased, the new solution is accepted with a probability depending on the energy variation and the current value of a *temperature* variable T, which tends to decrease along the process. The lower T is, the less likely it is that a worse solution will be accepted. The fact that worse solutions may be accepted allow the approach to explore more solutions, thus decreasing the possibility of becoming trapped in a local optimum. The process stops when a given stopping criterion is met. The corresponding pseudo code is presented in Algorithm 2.

In our case, the function $E()$ that computes the energy is the *Score()* function defined in Section 3.4. To initialize the variables (*variablesInitialization()*), we have just generated a random sampler.

To identify a new assignment of variables (*newAssignment(values)*), we increase the likelihood of disturbing the variables corresponding to roads carrying more routes among those involved in the most unsatisfied junctions. This favors the improvement of the most unsatisfied junctions; and choosing more often roads that carry more routes ensures better coverage of the solution space. More precisely, a road for which a new assignment

Algorithm 2 Pesudocode for simulated annealing approach to optimization

```
$T \leftarrow$ initialT
$variablesAssignment \leftarrow variablesInitialization()$
$energy \leftarrow E(variables)$
**while** $(NoStop())$
    $newVariablesAssignment \leftarrow newAssignment(variablesAssignment)$
    **if** $E(newVariablesAssignment) < energy$ **or**
    $random() < \exp((energy - E(newVariablesAssigment))/T)$ **then**
        $variablesAssignment \leftarrow newVariablesAssignment$
        $energy \leftarrow E(newVariablesAssignment)$
    $T \leftarrow$ lambda $* T$
**return** $variablesAssignment$
```

will be computed is identified as follow: we first choose a junction j with the probability *pJunction*(j); then, amongst the roads adjacent to the junction, we choose a road *rd* with a probability of $pRoad_j(rd)$. A new value is then randomly assigned to the x_{rd} variable associated to the chosen road *rd*. The probabilities are defined as follow:

$$pJunction(j) = \frac{f_{2_j}(X_j) + f_{3b_j}(X_j) + f_{4_j}(X_j)}{\sum_{j \in AllJunctionsWithRoutes}\left(f_{2_j}(X_j) + f_{3b_j}(X_j) + f_{4_j}(X_j)\right)}$$

$$pRoad_j(rd) = \frac{\#supported\_routes(rd)}{\sum_{rd\ adjacent\ to\ j} \#supported\_routes(rd)}$$

We stop the algorithm (*NoStop()* function to false) when the energy has been stagnating for a number of iterations ($20,000$ in our case).

We arbitrarily chose an initial temperature (initialT) and a lambda very close to 1. This way, we allow a wide enough exploration of the solutions space.

5 Implementation and results

We ran our methods on a real dataset composed of roads and hiking paths from BD TOPO, the 1m resolution database of IGN, and routes provided by FFRP (Fédération Française de Randonnée Pédestre), the French hiking federation, on which a conflation pre-process had been run to reach the modeling described in Section 3. The considered dataset covers an area of 10×20km. Given the density of roads and hiking routes, the dataset could be used to generate a map at a scale of about 1:50k. At this scale the test dataset has a size of 20×40cm (approximately an A3 leaflet). The test area assessed is representative of what can be found on real route maps in terms of complexity of route symbol placement (number of routes carried per road, complexity of junctions). Table 1 shows the distribution of roads of our test area in terms of number of carried routes, and for each number of carried routes, the cardinality of the variable domain. The total size of the search space, which is quite large, is shown in the bottom-right cell of the table.

We implemented our model in GeOxygene, an open source framework developed in Java at COGIT laboratory [3]. The main reason for choosing GeOxygene is that the opportunity to reuse existing generalization methods implemented in GeOxygene, more pre-

n	1	2	3	4	5	Total
Number of road strokes carrying n routes	48	29	24	6	10	117
Variables domains cardinality	2	6	24	125	720	$2^{48} \times 6^{29} \times \cdots \times 720^{10}$

Table 1: Test area description.

cisely in its generalization module CartAGen [16]. Regarding the work presented in this paper, we already had in GeOxygene an implementation of the static model for roads carrying routes, and an implemented method for roads strokes detection based on roads attributes and angles at junctions. We adapted this method by splitting the computed strokes if needed, so that all road segments of a stroke carry the same set of routes.

We generated in advance the possible domains D_n (described in Section 3.2.2) associated to the variables of our CSP. As a domain D_n for a variable associated to a given road only depends on the number of routes n carried by this road, we set an upper bound to n (in our case 5, consistent with our test data set) and generated the five domains D_1 to D_5 (i.e., $\{(-1);(1)\}$ for 1 route, $\{(-2,-1);(-1,-2);(-1,1);(1,-1);(1,2);(2,1)\}$ for 2 routes, etc.). Each domain D_n was defined as a java constant, consisting in a set of lists of n elements.

Once the CSP was created, we implemented the score functions for constraints as described in Sections 3.3 and 3.4, and implemented the backtracking and simulated annealing algorithms according to the pseudo codes shown in Algorithms 1 and 2.

5.1 Experimentations and assessment settings

The two implemented methods (systematic/backtracking, and non-systematic/simulated annealing) were run on a system powered by an Intel Core i5-2500 CPU (3.30 GHz) with 8GB of RAM. Both were assessed in terms of computational performance (evaluation 1, Section 5.2). Because of the computational performance issues already described, the systematic (backtracking) method could not be run on our complete test dataset. However, as it is guaranteed to reach the best possible solution, analyzing the cartographic results obtained using this method was a way to validate the defined set of constraints (evaluation 2, Section 5.3). Therefore the method was run on small extracts of our test dataset corresponding to complex zones—in other words, with respect to the CSP, it was run on restricted subsets of variables. The non-systematic (simulated annealing) method was run on the complete test area. It was executed 100 times to take into account it is initialized with randomized variables. The final scores obtained show few variations amongst the 100 executions, as well as the cartographic quality (visually assessed for a few results chosen randomly). Figure 18 shows an extract of the map corresponding to the best reached score. The cartographic results obtained were also evaluated (evaluation 3, Section 5.4). For evaluations 2 and 3, the cartographic results were shown to five cartographers working in production at IGN-France. Map series produced at IGN-France do not include such "bus-like" representation of routes, but IGN has regularly had to produce hiking (or cycling) maps using this representation for particular customers. Among the five cartographers to whom the results were shown, three directly took part to the production of those maps. The last two ones, including the maps product manager of IGN, have a good experience of hiking maps production and are in contact with maps users.

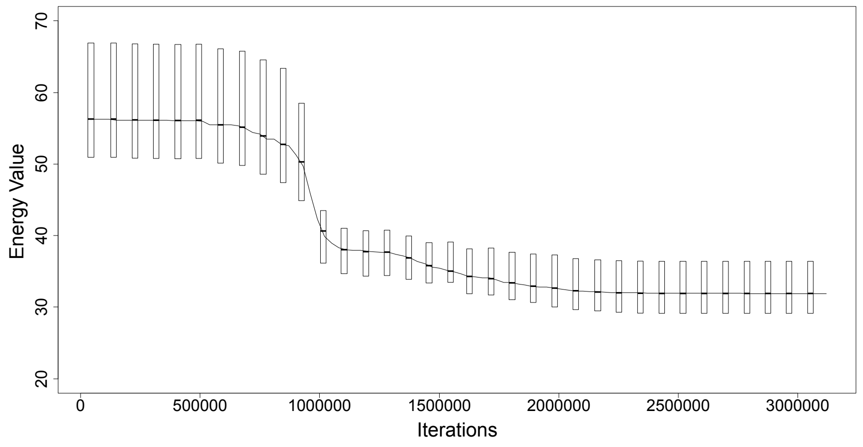

Figure 16: Convergence behavior of the simulated annealing process. The curve represents the median score value as a function of the number of iterations. The vertical bars show the first and third quartiles on the 100 executions. As all the runs do not stop after the same number of iterations, the energy value for the shorter runs were repeated after their end.

5.2 Computational performance assessment

The systematic (backtracking) algorithm described in Section 4.1 required 10 hours to find the optimal solution on a zone covering $1/10$ of the test area considered representative of the problem complexity. Knowing that the complexity of the algorithm is exponential in the number of variables, we drew the conclusion that it was not realistic to use it on the entire test map.

The simulated annealing method was run on the complete test area. The execution time until the process converges was relatively stable over the 100 executions: more or less 20 minutes, which is compatible with a use in production in the context of offline production of maps. Figure 16 shows the convergence behavior of the process. The results are consistent from an iteration to another. An important decreasing of energy value is observed at $900,000$ executions, which may be interpreted as the resolution of important conflicts.

5.3 Constraint set validation: Systematic (backtracking) method

Figure 17a shows an optimal placement obtained with the systematic (backtracking) method on a restricted road network. On another restricted road network, Figure 17 (b and c) shows two solutions assessed as optimal with different constraint parameters. On the left, the cost function used is the one presented in Sections 3.3 and 3.4. On the right, the weights of constraints have been slightly changed to favor the satisfaction of constraint C3b (number of route intersections at junctions) over constraint C2 (route continuity).

These three extracts, enlarged to a bigger scale to be easily analyzed, were sent to the cartographers and then discussed with them. Indeed, on Figure 17 (b and c) the vertical road segment in the middle has been shortened for the needs of the paper, while the original extract was provided to them (still for the needs of the paper, the two zones shown in

Figure 17 have been highlighted on Figure 18). No consensus could be drawn from the reactions of the cartographers—both solutions were assessed as good, and none was clearly preferred.

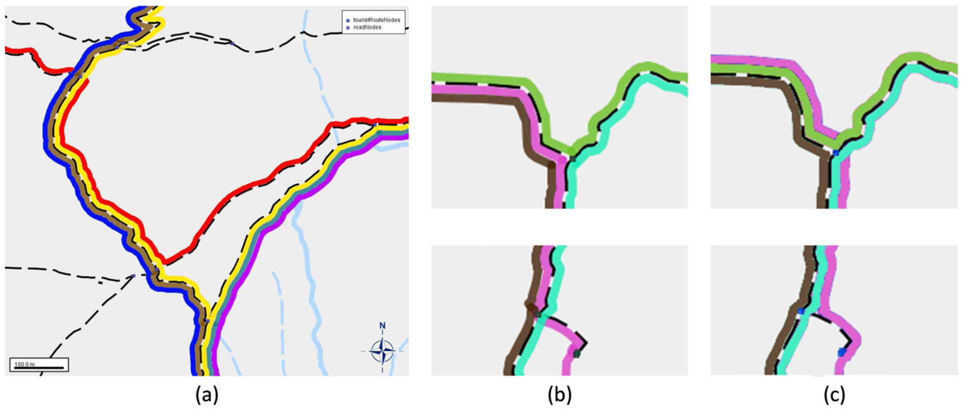

Figure 17: Optimal solution obtained on complex zones.

5.4 Assessment of non-systematic (simulated annealing) results

The map extract shown in Figure 18[3] was separately provided to two groups of cartographers, together with the extracts shown in Figure 17: on the one hand the group who previously took part to the production of "bus-like" route maps, and on the other hand the other two cartographers. This map is the one with the best score among the 100 executions of the simulated annealing process. The extract covers a surface of approximately 1/3 of the total test zone, and has been chosen because it is the most complex and therefore the most interesting part of the test zone in terms of route symbol placement. An SVG file of the extract was sent to the cartographers so that they could print it at real scale on the plotter they usually use to assess proof maps in production. We did not set up a formal survey as, for example, in [23]. Instead we asked them to assess the route placement, and asked them for comments on what could be improved. We had separate informal interviews with each of the two groups the day after, so that they could spend time to study the results. The two groups considered the placement was good in general. The group of cartographers who already had to manually produce such maps considered the cartographic quality of the placement comparable to what they would have done manually (it is worth underlining, after our experience, that traditional cartographers used to working manually are often upset by the idea that a machine could be as effective as themselves). The cartographers pointed some defects of our drawing method (which we were already aware of), and a few places where the presence of a close neighboring road should have led to placing the route symbols on the opposite side of the road. They also highlighted that the color contrasts between adjacent routes should be improved.

[3]Here, the two zones used to assess the optimal solution obtained with the systematic (backtracking) method have been highlighted, which was not the case in the version sent to the cartographers.

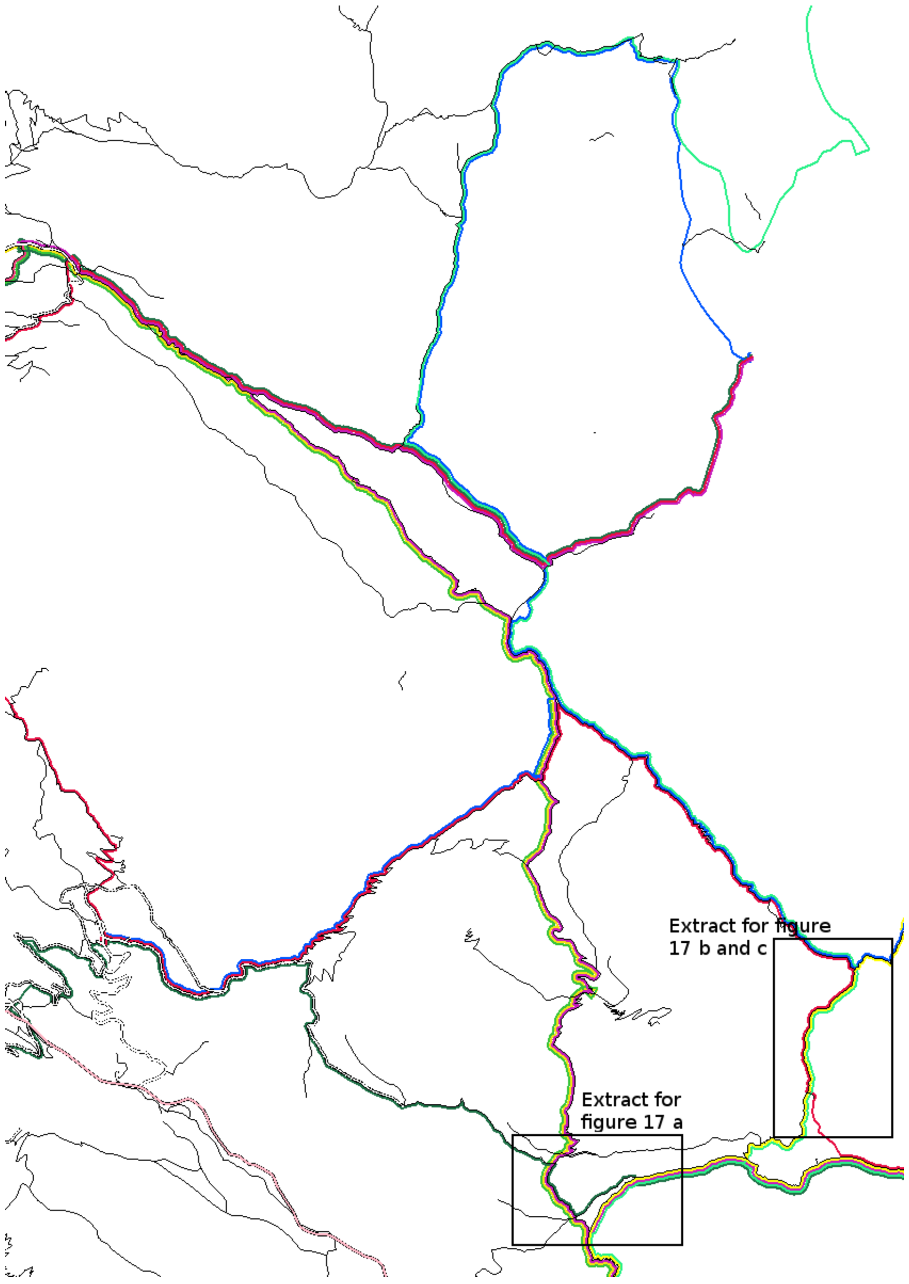

Figure 18: Best solution reached by the simulated annealing process. Extract representing 1/3 of the area of the test zone, the most complex in terms of route symbols to place.

6 Conclusion and perspectives

In the study presented in this paper, we identified a set of constraints for a problem that, to our knowledge, had not been tackled until now: multiple route symbol placement along a road network. We proposed modeling of this problem as a constraint optimization problem. Moreover, we suggested two different solving methods. A systematic search with a backtracking algorithm enabled the validation of our constraint system. A non-systematic search with a simulated annealing algorithm gave very good results on a real case study that is representative of relatively complex situations.

6.1 Possible improvements

We identify four different categories of possible improvements to our proposal. All these perspectives will have to be explored in future works.

First, we use a perfectible drawing method that generates some bad looking junctions even when routes are correctly placed (cf. Section 3.2.1). Working on a drawing method may be a first angle to improve our graphical results. This method should take into account our modeling of the route positions along a road.

Then, some aspects of the problem were ignored as described in the problem statement Section 3, in order to first validate the approach globally. These aspects are:

- Angles: The modification of the position of a road may have a different impact depending of the angle, as we may observe on Figure 5, where Figure 5(b) seems better than Figure 5(a).
- Colors: As pointed out by the cartographers, having adjacent routes with poor color contrast is a problem. This could be remedied while including color contrast constraints in our model. Existing studies about color contrasts, such as [4] or [10], could be used as a basis.
- Hierarchy between routes: In some application contexts a hierarchy between routes has to be shown, e.g., to highlight a particular hiking route, or in the case of routes that would represent flows. A hierarchy of color values, dash styles, or line widths could be used, but this results in new additional constraints, e.g., it should be avoided to place one narrow route between two larger routes.

A third way to enhance our model and the results produced concerns the solving method. One identified limit of our simulated annealing method is that some simple zones, typically dead ends of the route graph at the borders of the test area, are not always optimally handled because the corresponding variables are not disturbed often enough. This is a drawback of our method to generate neighboring solutions from a current state, which favors complex zones. To cope with that, a systematic search with the backtracking algorithm could be run on such simple areas after the simulated annealing has finished. A first test giving good results was carried in that direction. However it would be necessary to automatically identify those simple zones. Likewise, in order to improve the performance of our simulated annealing method, we could split the problem into sub problems and solve them in sequence while taking already computed results as external constraints. For that, sub-areas of the map could be identified e.g., thanks to the isthmuses (in the graph theory acceptation of the word) of the road network. However, this partitioning may not always be possible.

Finally, adding route symbols along roads requires space on a map. Thus, it could impact the symbolization and the generalization of surroundings objects map (buildings, waterlines, water areas, etc.). These modifications have to be done while considering the fact that the positions of routes on roads may change. This implies the use of methods taking into account the local management of geographic objects. This is the topic of a case study within an ongoing research project, dealing with interactions between geographic objects in an automated generalization model.

6.2 Reuse for other application cases

Another perspective concerns the application of our method to other kinds of routes and physical network infrastructures, different from hiking routes on a road network. As mentioned in the introduction, we think that our method can be used for a diversity of application cases. We think that the constraints defined in the paper are likely to be valid for most situations, as they model continuity aspects that help perceive a route as a whole. However, more constraints might be needed depending on specific characteristics of other application cases. For instance, in the case of electric cables sharing conducts, a more complex, multi-level modeling may be needed [13]. Besides, one specific characteristic of the hiking routes application case is that no physical hubs that would anyway disrupt the continuity of routes are displayed on the map at junctions—contrary to what happens, for example, on airways routes, where an airport (hub) is displayed at every junction where two routes begin or end to share the same physical network segment [10]. In other application cases, hubs can be present only at some junctions, e.g., in the case of multimodal freight transport routes. It would be interesting to study how the presence of hubs at junctions influences the importance of the route continuity constraints at these junctions. Are they all the more important as the hub tends to break the continuity, or can they be relaxed since the hub tends to break it anyway?

Acknowledgements

The authors would like to thank Anne Bérard, Didier Blaquière, Régine Cochet, Blandine Delavault and Christophe Pécot for their time and the interesting discussions around the cartographic results assessment. They also would like to thank Eve Feinblatt-Mélèze for her copy editing work. They finally would like to thank the anonymous reviewers who helped improve the paper by their thorough comments and questions.

References

[1] BEARD, K. M. Constraints on rule formation. In *Map Generalization: Making Decisions for Knowledge Representation*, B. Buttenfield and R. Mcmaster, Eds. Longman Scientific and Technical, London, 1990, pp. 121–135.

[2] BELBIN, J. Gestalt theory applied to cartographic text. In *Cartographic Design: Theoretical and Practical Perspectives*, C. H. Wood and C. P. Keller, Eds. Wiley-Blackwell, 1996, pp. 235–269. ISBN13: 9780471965879.

[3] BUCHER, B., BRASEBIN, M., BUARD, E., GROSSO, E., MUSTIÈRE, S., AND PERRET, J. Geoxygene: Built on top of the expertise of the french nma to host and share advanced gi science research results. In *Geospatial Free and Open Source Software in the 21st Century*, E. Bocher and M. Neteler, Eds., Lecture Notes in Geoinformation and Cartography. Springer, Berlin, 2012, pp. 21–33. doi:10.1007/978-3-642-10595-1_2.

[4] CHESNEAU, E., RUAS, A., AND BONIN, O. Colour contrasts analysis for a better legibility of graphic signs on risk maps. In *Proc. 22th International Cartographic Conference (ICC)* (La Coruña (Spain), 2005).

[5] CHRISTENSEN, J., SHIEBER, S., AND MARKS, J. Placing text labels on maps and diagrams. In *Graphics Gems*, P. S. Heckbert, Ed. Academic Press, 1994, pp. 497–504. doi:10.1016/B978-0-12-336156-1.50064-1.

[6] DE LUCIA, A., AND BLACK, T. A. A comprehensive approach to automatic feature generalization. In *Proc. 13th International Cartographic Conference (ICC)* (Morelia (Mexico), july 1987), pp. 168–191.

[7] FINK, M., AND PUPYREV, S. Metro-line crossing minimization: Hardness, approximations, and tractable cases. In *Graph Drawing*, S. Wismath and A. Wolff, Eds., vol. 8242 of *Lecture Notes in Computer Science*. Springer, Berlin, 2013, pp. 328–339. doi:10.1007/978-3-319-03841-4_29.

[8] FINK, M., AND PUPYREV, S. Ordering metro lines by block crossings. In *Mathematical Foundations of Computer Science 2013*, K. Chatterjee and J. Sgall, Eds., vol. 8087 of *Lecture Notes in Computer Science*. Springer, Berlin, 2013, pp. 397–408. doi:10.1007/978-3-642-40313-2_36.

[9] FREUDER, E. C., AND MACKWORTH, A. K. Constraint satisfaction: An emerging paradigm. In *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., vol. 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006, pp. 13–27. doi:10.1016/S1574-6526(06)80006-4.

[10] HURTER, C., SERRURIER, M., ALONSO, R., TABART, G., AND VINOT, J.-L. An automatic generation of schematic maps to display flight routes for air traffic controllers: Structure and color optimization. In *Proc. International Conference on Advanced Visual Interfaces (AVI)* (New York, NY, USA, 2010), ACM, pp. 233–240. doi:10.1145/1842993.1843034.

[11] KIRKPATRICK, S., GELATT JR., C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680. doi:10.1126/science.220.4598.671.

[12] LI, Z., AND ZHOU, Q. Integration of linear and areal hierarchies for continuous multiscale representation of road networks. *International Journal of Geographical Information Science 26*, 5 (2012), 855–880. doi:10.1080/13658816.2011.616861.

[13] LÜSCHER, P., STOOP, P., FEHLMANN, J., AND SPARENBORG, J. Automatic generation of network schematics from cadastral databases: Applications for an electric utility company. In *Proc. 15th ICA Workshop on Generalisation and Multiple Representation* (2012).

[14] NÖLLENBURG, M. A survey on automated metro map layout methods. Tech. rep., Department of Informatics, Karlsruhe Institute of Technology (KIT), 2014.

[15] OKSANEN, J., SCHWARZBACH, F., SARJAKOSKI, L. T., AND SARJAKOSKI, T. Map design for a multi-publishing framework: Case menomaps in nuuksio national park. *The Cartographic Journal 48*, 2 (2011), 116–123. doi:10.1179/1743277411Y.0000000007.

[16] RENARD, J., GAFFURI, J., AND DUCÊNE, C. Capitalisation problem in research—Example of a new platform for generalisation: CartAGen. In *Proc. 13th ICA Workshop on Generalisation and Multiple Representation* (sept 2010).

[17] SMITH, B. Modelling. In *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., vol. 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006, pp. 375–404. doi:10.1016/S1574-6526(06)80015-5.

[18] THOMSON, R., AND RICHARDSON, D. The "good continuation" principle of perceptual organization applied to the generalization of road networks. In *Proc. 19th International Cartographic Conference (ICC)* (Ottawa (Canada), 1999).

[19] TOUYA, G. A road network selection process based on data enrichment and structure detection. *Transactions in GIS 14*, 5 (2010), 595–614. doi:10.1111/j.1467-9671.2010.01215.x.

[20] VAN BEEK, P. Backtracking search algorithms. In *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., vol. 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006, pp. 83–132. doi:10.1016/S1574-6526(06)80008-8.

[21] VAN HOEVE, W., AND KTRIEL, I. Global constraints. In *Handbook of Constraint Programming*, P. v. B. Francesca Rossi and T. Walsh, Eds., vol. 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006, pp. 205–244. doi:10.1016/S1574-6526(06)80010-6.

[22] WARE, J. M., AND JONES, C. B. Conflict reduction in map generalization using iterative improvement. *GeoInformatica 2*, 4 (1998), 383–407.

[23] WEISS, R., AND WEIBEL, R. Road network selection for small-scale maps using an improved centrality-based algorithm. *Journal of Spatial Information Science 2014*, 9 (2014), 71–99. doi:10.5311/JOSIS.2014.9.166.

[24] WERTHEIMER, M. Untersuchungen zur Lehre von der Gestalt. II. *Psychologische Forschung 4*, 1 (1923), 301–350. doi:10.1007/BF00410640.