Spatial Aggregation: Language and Applications*

Christopher Bailey-Kellogg and **Feng Zhao**
Computer and Information Science Department
The Ohio State University
2015 Neil Ave.
Columbus, OH 43210 U.S.A.
{kellogg,fz}@cis.ohio-state.edu

Kenneth Yip
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139 U.S.A.
yip@martigny.ai.mit.edu

Abstract

Spatial aggregation is a framework for organizing computations around image-like, analogue representations of physical processes in data interpretation and control tasks. It conceptualizes common computational structures in a class of implemented problem solvers for difficult scientific and engineering problems. It comprises a mechanism, a language, and a programming style. The spatial aggregation mechanism transforms a numerical input field to successively higher-level descriptions by applying a small, identical set of operators to each layer given a metric, neighborhood relation and equivalence relation. This paper describes the spatial aggregation language and its applications.

The spatial aggregation language provides two abstract data types — neighborhood graph and field — and a set of interface operators for constructing the transformations of the field, together with a library of component implementations from which a user can mix-and-match and specialize for a particular application. The language allows users to isolate and express important computational ideas in different problem domains while hiding low-level details. We illustrate the use of the language with examples ranging from trajectory grouping in dynamics interpretation to region growing in image analysis. Programs for these different task domains can be written in a modular, concise fashion in the spatial aggregation language.

Introduction

Effective reasoning about a physical system requires an appropriate mapping from the system characteristics to abstractions that match the requirements of the task at hand. Spatial aggregation organizes computations around image-like, analogue representations of physical processes in data interpretation and control tasks (Yip & Zhao 1996). In Qualitative Physics, three ontological abstractions are widely used: device, process, and constraint. Spatial aggregation introduces a new ontological abstraction, the *field ontology*, to unify many reasoning tasks involving the image-like analogue representations such as the velocity field for fluid motion, phase space for dynamical systems, and configuration space for mechanism analysis.

The input to spatial aggregation is a data massive, numerical field. [1] The desired output is a high-level, parsimonious description of the structure and behavior of the physical process that the field represents. To bridge the semantic gap between the analogue input field and the final symbolic description, spatial aggregation introduces layers of intermediate structures called spatial aggregates to capture spatial adjacencies among objects of the field at multiple spatial and temporal scales. A spatial aggregate is constructed from a metric, a neighborhood relation and an equivalence relation supplied by a user according to the objective of computation. The spatial aggregation mechanism transforms the input field to successively higher-level descriptions by applying a small, identical set of operators to each layer of the spatial aggregates.

The spatial aggregation framework grows out of a class of problem solvers, KAM (Yip 1991), MAPS (Zhao 1994) and HIPAIR (Joskowicz & Sacks 1991), that derive their power primarily from perceptual operators on analogue representations, and only secondarily from search and analytical methods. These programs have exhibited expert performance on difficult problems in hydrodynamics, nonlinear control, and engineering mechanism analysis. Spatial aggregation abstracts the common computational structure and a set of generic operators from these problem solvers. It can also apply to a wide variety of other task domains such as image analysis and geographic information databases applications. The generic op-

* FZ is supported by an NSF National Young Investigator Award CCR-9457802, a Sloan Foundation Research Fellowship, a grant from Xerox Palo Alto Research Center, and an NSF grant CCR-9308639. CBK is supported by FZ's NSF NYI grant CCR-9457802. KY is supported by an NSF National Young Investigator Award ECS-935777.

[1] A field maps one continuum to another. Examples include velocity field ($R^3 \rightarrow R^3$), temperature field ($R^3 \rightarrow R^1$), image field ($R^2 \rightarrow R^1$), and vector field ($R^n \rightarrow R^n$).

erators of spatial aggregation can be viewed as a particular instantiation of Ullman's "visual routines" for visual information processing tasks (Ullman 1984; Mahoney 1995).

Other researchers have developed related frameworks and systems for reasoning about spatial, analogue representations of physical world. For example, Forbus et al. developed the Metric Diagram/Place Vocabulary (MD/PV) framework for qualitative spatial reasoning (Forbus, Nielsen, & Faltings 1991). Chandrasekaran and Narayanan proposed a direct analogue simulation of elementary mechanics problem using a diagrammatic representation (Chandrasekaran & Narayanan 1990). In comparison, the spatial aggregation framework comprises multi-layer spatial aggregates with identical computational structure at each layer and focuses on the problem of recovering structures from numerical fields.

This paper describes the spatial aggregation language and the development of applications in the style of spatial aggregation. We give an overview of the basic features of the language, illustrated with an example of trajectory grouping in dynamics analysis. We describe the language syntax, component library, implementation, and a sample program written in the language for a region growing operation in image analysis. We then summarize the language experience in application development.

Overview

Given an input field, the spatial aggregation mechanism constructs a neighborhood graph (N-graph) from primitive objects of the field, explicates their spatial adjacencies, and forms equivalence classes of these objects using an equivalence relation determined by the objective of the task. An equivalence class can be redescribed as a primitive object at a higher level if necessary, and the identical steps of aggregation to form a new N-graph and classification to form equivalence classes apply with a new metric, neighborhood relation, and equivalence relation. This iteration terminates when the desired behavioral and structural description can be readily derived from the N-graph. The N-graph and field serve as computational glue for the operations that search, transform, and filter the spatial objects. Figure 1 illustrates the data flow in the main operations of the language at each level of spatial aggregation.

Spatial aggregation represents primitive objects of a physical process or system with *spatial objects*. For instance, a spatial object might describe a state of a dynamical system — a point and its direction of movement in an n-dimensional phase space spanned by the

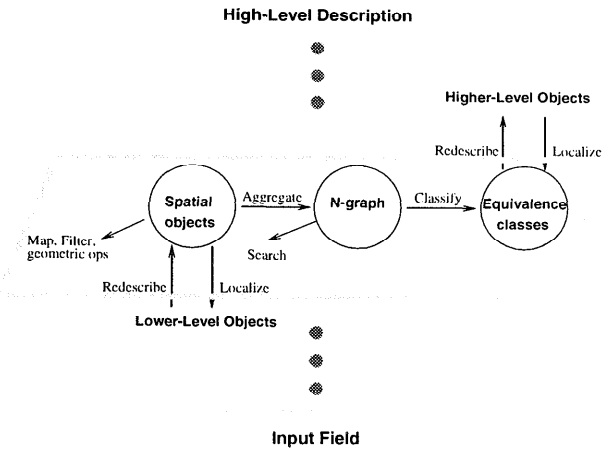


Figure 1: Spatial aggregation: the lower-level objects from the numerical input field are transformed into higher-level objects through a sequence of operations available in the language. The higher-level objects then become the input to another level of spatial aggregation where the identical set of operators apply.

state variables. A spatial object comprises a geometric description and a feature description. The geometric description is specified in a *metric space* defining distances between geometric primitives. The feature description belongs to one or more *feature spaces*. For example, in image analysis, a pixel spatial object uses the pixel location as the geometric description and the associated brightness value as the feature description. Likewise, a region spatial object defines a geometric region in an image and an average or minimum/maximum brightness value of the constituent pixels. In a meteorology application, each spatial object specifies a location in space and a temperature, barometric pressure, and air flow velocity in feature space. The distance between values in a feature space represents how different the corresponding spatial objects are. Spatial aggregation forms neighborhood graphs for spatial objects using the geometric description and groups the spatial objects using similarity or proximity measures in feature space. For example, spatial aggregation could group text with the same font, using a feature space defined by font characteristics. In a mechanism analysis system, it could group configurations in a configuration space.

As an example of how spatial aggregation provides organizational principles and building blocks to facilitate the development of programs for engineering problems, consider an interpretation task in dynamical sys-

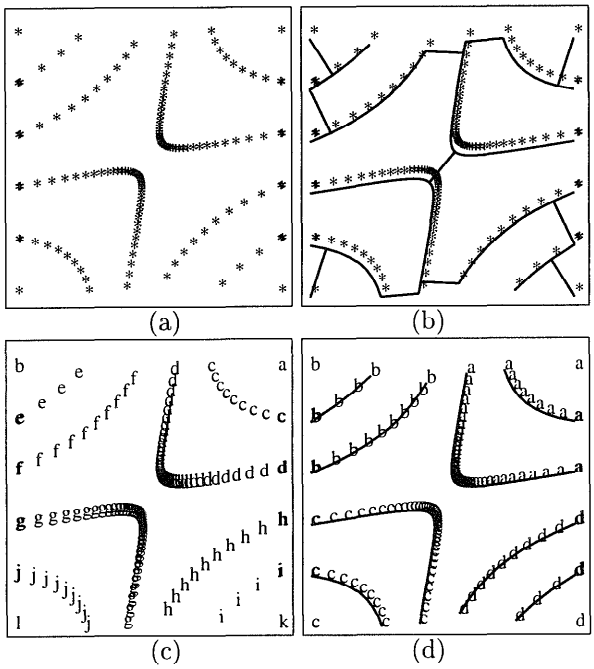


Figure 2: Trajectory interpretation: (a) Input field: points in phase space. (b) A neighborhood graph (MST) for the points. (c) Points grouped into trajectories labeled by a through l respectively. (d) Trajectories grouped into bundles labeled a through d respectively.

```
(define points-ngraph
  (aggregate input-field points-ngraph-fac))
```

(a)

```
(define point-classes
  (classify
   points-ngraph points *point-distance-tol*))
```

(b)

```
(define trajs
  (redescribe point-classes traj/create))
```

(c)

```
;; Aggregate the trajectories.
(define traj-ngraph
  (aggregate trajs traj-ngraph-fac))

;; Form equivalence classes.
(define traj-bundles
  (classify
   traj-ngraph trajs *vector-similarity-tol*))
```

(d)

Figure 3: Trajectory interpretation code: (a) Aggregation of trajectory points. (b) Classification of trajectory points. (c) Redescription of point classes as trajectories. (d) Aggregation and classification of trajectories.

tem analysis. The input is a field of sampled states as points in phase space shown in Figure 2(a). The objective is to group the states into trajectories and then trajectories into trajectory bundles that share the same qualitative behaviors, as shown in Figure 2(c) and Figure 2(d) respectively. Even though the example is in 2D space, the operators apply to higher-dimensional spaces as well.

The first step, *aggregation*, forms a neighborhood graph using a neighborhood relation to explicitly indicate pairs of adjacent spatial objects. Different applications require different neighborhood relations. In the trajectory interpretation application, a minimal spanning tree (MST) is appropriate; other applications use Voronoi diagrams, nearness criteria, and so forth. The spatial aggregation code shown in Figure 3(a) uses the operator `aggregate` to compute the neighborhood graph (the argument `points-ngraph-fac` — constructed from language library components — specifies how to build an MST). Figure 2(b) shows the result. The operator `aggregate` allows the user to focus on choosing a good neighborhood relation while hiding implementation details.

The next main step, *classification*, forms equivalence classes of neighboring spatial objects according to their similarity in the feature space. In the trajectory interpretation example, a point can be considered similar to a neighbor if their separation is not significantly longer than the distances separating other nearby neighbors. In the code of Figure 3(b), `classify` forms equivalence classes of points from the MST, `points-ngraph`, by deleting edges that are too long according to the threshold `*point-distance-tol*`; the result is shown in Figure 2(c). Other classifiers, discussed later in the paper, use more powerful mechanisms, such as consistency predicates testing formed classes. The operator `classify` allows the user to select an appropriate equivalence relation and a classification mechanism.

The third main step of spatial aggregation, *redescribing*, maps equivalence classes of objects at one level to single higher-level objects at the next level. In the trajectory interpretation example, each equivalence class of points becomes a single trajectory object; Figure 3(c) shows the spatial aggregation code. The `redescribe` operator shifts the level of abstraction so that the aggregation process can repeat at a higher level. The inverse of redescribing is *localizing*, which maps each higher-level object to the equivalence class of constituent objects at the lower level.

To group trajectories into trajectory bundles, the

same process repeats, using the operators `aggregate`, `classify`, and `redescribe`. The only differences are in the metric, neighborhood relation, and equivalence relation: trajectories are aggregated into a neighborhood graph where the neighborhood is defined by a sphere of some fixed radius, and neighboring trajectories are bundled using an equivalence relation comparing corresponding vectors along trajectories. Figure 3(d) shows the spatial aggregation code, and Figure 2(d) shows the result. The aggregation process can repeat at a even higher level if necessary.

As the example illustrates, programs written in the spatial aggregation language are modular, using a common data structure (neighborhood graph) and an identical set of generic operators. They are concise and make explicit the important computational characteristics of the problem: neighborhood and equivalence relations.

Additional operators are available for manipulating the objects in the neighborhood graph. For example, *search* starts at any of a list of objects in the graph and moves from neighbor to neighbor, following some desired control strategy (e.g. depth-first search or breadth-first search) and finding paths satisfying some criteria. Interfaces to standard geometric and numerical libraries could further extend the capabilities of the language.

Spatial Aggregation Language

The spatial aggregation language provides operators and abstract data types (ADTs), together with a library of basic components providing commonly used implementations, for constructing the transformations of the field. To use the language, a user selects operators and components, mixing-and-matching and specializing them with necessary field metric and similarity measure information for each spatial aggregate layer.

The core of the language

The core of the language comprises two abstract data types, the `field` and the `ngraph`, and a set of interface operators. Table 1 summarizes the syntax of the language data types and operators.

Field A `field` defines a metric space for the geometric descriptions of spatial objects, and can answer spatial queries. The `field` data type is supported in the language component library by several commonly used spatial indexing methods (e.g. array, grid, and k-d tree).

N-graph An `ngraph` defines a neighborhood relation for a set of spatial objects, and can return the neighbors of any given object defined in the `ngraph`. A wide variety of `ngraph` implementations, available in the component library, support different neighborhood relations such as nearness, MST, and Voronoi diagram.

- Data types:
  - `Field` interface:
    create: objects → field
    Returns a `field` indexing the `objects`.
    domain: field → objects
    Returns `objects` defined in the `field`.
    near: field * object * distance → objects
    Returns `objects` within `distance` of the `object` in the `field`.
  - `Ngraph` interface:
    create: field → ngraph
    Returns an `ngraph` aggregating objects in the `field`.
    neighbors: ngraph * object → objects
    Returns neighbors of the `object` in the `ngraph`.
- Interface operators:

  aggregate: field * ngraph-fac → ngraph
  Groups objects of the `field` into an `ngraph`.

  classify: ngraph * objects * threshold → classes
  Returns equivalence `classes` of the `objects` in the `ngraph` according to the feature similarity `threshold`.

  redescribe: classes * redescribe-function → objects
  Abstracts equivalence `classes` to higher-level `objects`.

  localize: objects * localize-function → classes
  Converts higher-level `objects` to `classes` of constituent objects.

  search: ngraph * objects * goal-predicate → paths
  Returns `paths` through the `ngraph` starting from the `objects` and satisfying `goal-predicate`.

  map, **filter**, and geometric operations on spatial objects.

  A user must specify the neighborhood relation, field metric, and equivalence relation for these operators explicitly, or provide procedures that compute the `ngraph` *and equivalence classes.*

Table 1: Syntax of the spatial aggregation language.

Interface Operations The `ngraph` and `field` are constructed and accessed by a set of language interface operators defined in Table 1.

Component library

A prototype of the language is implemented in Scheme. The ADTs are highly parameterized and can be instantiated with particular field metrics, similarity measures, etc. Some of the ADTs are pre-specialized with commonly-used values for efficiency reasons. The modular design of the ADTs supports language extensions and user control over tradeoffs such as efficiency vs.

- `Field`:

  Spatial indices: **Array**, **grid**, **k-d tree**, **list**.

  Intensional: User-specified function defines field objects.

  Interpolated: User-specified function interpolates field objects from given values.

- `Ngraph` neighborhoods:

  Graphical methods: **MST**, **Voronoi diagram**.

  Nearness: Neighbors within specified radius.

  Generating function: Neighbors defined by user-specified function.

- `Classify` mechanisms:

  Standard: Objects are considered equivalent if they are neighbors and similar in feature space; equivalence classes are formed by transitive closure.

  Splitting: Classifies with a loose threshold and then applies a user-supplied consistency check to the classes. Reclassifies inconsistent classes with successively tighter thresholds.

  Merging: Classifies with a tight threshold and then merges similar classes using a higher-level aggregation process.

  Stabilizing: Compares classifications over a range of thresholds, returning the one that persists over the largest subrange.

Table 2: Library of commonly used components in the spatial aggregation language.

generality (Weide, Ogden, & Zweben 1991).

The current component library, shown in Table 2, contains basic implementations for each abstract data type and operator. Other component implementations can be built according to the defined specifications and added to the library if necessary.

Example program

We use a region-growing example from image analysis to illustrate how a simple program can be written using the language (Figure 4). The program takes as input a field — an image mapping pixel coordinates to brightness values (Figure 5(a)) — and produces a list of disjoint regions of pixels with similar brightness values (Figure 5(b)). The pixel spatial objects comprise points in a subspace of R^2 and corresponding gray-scale values from $\{0, 1, ..., 255\}$.

Language Experience

The spatial aggregation language is applicable to a wide range of problem domains, including dynamical system analysis, fluid flow motion analysis, me-

```
;; A 14x14 array field
(define image-field-fac
  (field-array/instantiate '(14 14) pixel/point))
```

(a)

```
;; 4-adjacency neighborhood using nearness ngraph
;; Neighbors are pixels one unit away
(define image-ngraph-fac
  (ngraph-near/instantiate image-field-fac 1))
```

(b)

```
;; Standard classifier
;; Similarity measure uses pixel values
(define classify
  (classify-standard/instantiate
   image-ngraph-fac
   (lambda (n1 n2)
    (abs (- (pixel/value n1)
            (pixel/value n2))))))
```

(c)

```
;; Form a neighborhood graph
(define image-ngraph
  (aggregate pixels image-ngraph-fac))
```

(d)

```
;; Form equivalence classes
(define classes
  (classify image-ngraph pixels *thresh*))
```

(e)

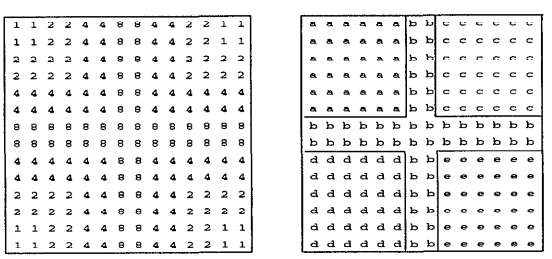
```
;; Create region objects from lists of pixels
(define regions
  (redescribe classes region/create))
```

(f)

```
;; Return lists of pixels from region objects
(define classes
  (localize regions region/pixels))
```

(g)

Figure 4: Region growing code: (a) Field instantiation. (b) N-graph instantiation. (c) Classifier instantiation. (d) `aggregate`. (e) `classify`. (f) `redescribe` pixel equivalence classes as regions. (g) `localize`.



(a) (b)

Figure 5: An example of region growing: (a) Input image: A 14×14 array of pixels. (b) Output image: five regions, a through e, of pixels of similar brightness.

chanical mechanism analysis, image analysis, auditory scene analysis, data mining, and geographic information databases. We have developed several small-scale application programs written in this language. Based on our experience, programming in the spatial aggregation language has several advantages:

1. The language allows a user to isolate what is important and express the important computational ideas in terms of the formation of equivalence classes and the transformation of neighborhood graphs, while hiding low-level implementation details. For example, the `classify` operator provides means for a user to specify and search for appropriate classification thresholds. The resulting programs are modular and concise.

2. The language provides field and N-graph data types for naturally representing physical objects in continuous domains. Field is a commonly used abstraction in science and engineering and hence facilitates the scientific and engineering applications of the language. N-graph serves as a common interface for developing programs. The interface operators are identical for different layers of spatial aggregation.

3. For a given task, a user can craft a program by mixing and matching and specializing components from the library provided by the language. A user has fine control over efficiency and generality in the language implementation and can extend the language capability by adding additional component implementations. Specializing data types through partial instantiation can improve performance; so can a more efficient implementation of a component. For example, a k-d tree field facility that replaces a grid can improve the object indexing performance in manipulating non-uniformly distributed points.

The current implementation of the language is limited in a number of ways. We plan to incorporate additional types of components, provide additional component implementations, and improve computational efficiency of the implementation. Other goals include the implementation of lazy evaluation and incremental analysis and update for N-graphs. To apply the language to large-scale problems, we need to build interfaces to existing numerical and computational geometry libraries so that the language can tap the power of the existing software base.

Conclusion

We have described an implemented language that supports programming in the style of spatial aggregation for a number of applications ranging from dynamics interpretation to image analysis. The spatial aggregation language provides primitives — field, N-graph, and a small set of operators — and means of abstraction for building problem solvers that derive concise symbolic descriptions from analogue representations of physical phenomena. Our experience provides evidence that the language supports the development of modular programs at an appropriate level of abstraction.

A central problem in artificial intelligence is to understand and construct the mappings from analogue signals to symbols and back. Spatial aggregation achieves a descriptive economy for an analogue input field by successively forming equivalence classes of lower-level objects and transforming a multi-layer of spatial aggregates, and is a possible realization of the signal-to-symbol mapping. Many important research questions remain open: What class of scientific problems can be formulated and solved in the style of spatial aggregation? Is there biological evidence that the brain might be performing spatial aggregation? What are other styles of reasoning that might bridge the analogue signals with the symbols?

References

Chandrasekaran, B., and Narayanan, N. 1990. Towards a theory of commonsense visual reasoning. In Nori, K., and Madhavan, C., eds., *Foundations of Software Technology and Theoretical Computer Science.* Springer.

Forbus, K.; Nielsen, P.; and Faltings, B. 1991. Qualitative spatial reasoning: the CLOCK project. *Artificial Intelligence* 51.

Joskowicz, L., and Sacks, E. 1991. Computational kinematics. *Artificial Intelligence* 51:381–416.

Mahoney, J. 1995. Signal-based figure/ground separation. Preprint.

Ullman, S. 1984. Visual routines. *Cognition* 18.

Weide, B.; Ogden, W.; and Zweben, S. 1991. Reusable software components. *Advances in Computers* 33:1–65.

Yip, K. M., and Zhao, F. 1996. Spatial aggregation: Theory and applications. *J. Artificial Intelligence Research.* To appear. Available from `http://www.cis.ohio-state.edu/~fz/`.

Yip, K. M. 1991. *KAM: A system for intelligently guiding numerical experimentation by computer.* MIT Press.

Zhao, F. 1994. Extracting and representing qualitative behaviors of complex systems in phase spaces. *Artificial Intelligence* 69(1-2):51–92.