Matching 11

11.1 ASPECTS OF MATCHING

11.1.1 Interpretation: Construction, Matching, and Labeling

Figure 10.1 shows a vision system organization in which there are several representations for visual entities. A complex vision system will at any time have several coexisting representations for visual inputs and other knowledge. Perception is the process of integrating the visual input with the preexisting representations, for whatever purpose. Recognition, belief maintenance, goalseeking, or building complex descriptions—all involve forming or finding relations between internal representations. These correspondences *match* ("model," "re-represent," "abstract," "label") entities at one level with those at another level.

Ultimately, matching "establishes an interpretation" of input data, where an interpretation is the correspondence between models represented in a computer and the external world of phenomena and objects. To do this, matching associates different representations, hence establishing a connection between their interpretations in the world. Figure 11.1 illustrates this point. Matching associates TOKNODE, a token for a linear geometric structure derived from image segmentation efforts with a model token NODE101 for a particular road. The token TOKNODE has the interpretation of an image entity; NODE101 has the interpretation of a particular road.

One way to relate representations is to *construct* one from the other. An example is the construction of an intrinsic image from raw visual input. Bottom-up construction in a complex visual system is for reliably useful, domain-independent, goal-independent processing steps. Such steps rely only on "compiled-in" ("hard-wired," "innate") knowledge supplied by the designer of the system. Matching becomes more important as the needed processing becomes more diverse and idiosyncratic to an individual's experience, goals, and

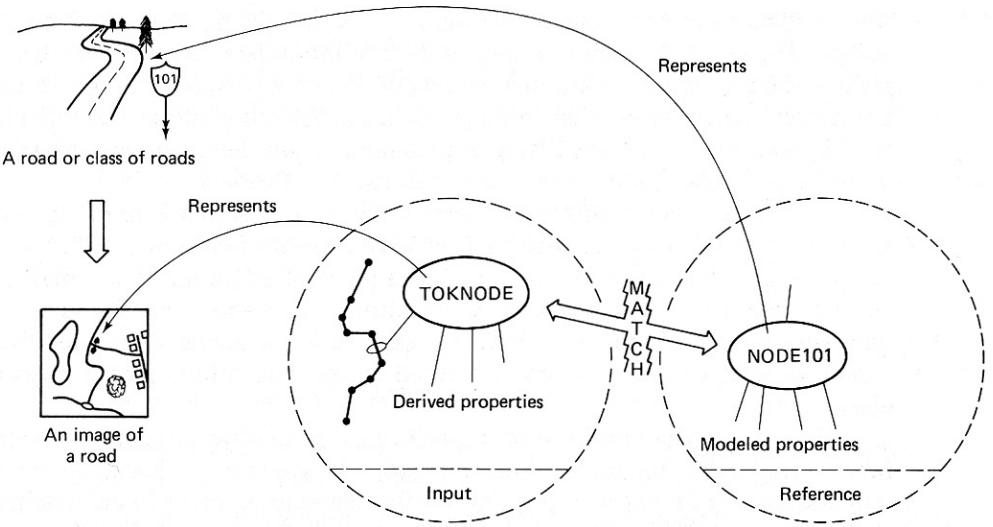


Fig. 11.1 Matching and interpretation.

knowledge. Thus as processing moves from "early" to "late," control shifts from bottom-up toward top-down, and existing knowledge begins to dominate perception.

This chapter deals with some aspects of matching, in which two already existing representations are put into correspondence. When the two representations are similar (both are images or relational structures, say), "matching" can be used in its familiar sense. When the representations are different (one image and one geometric structure, say), we use "matching" in an extended sense; perhaps "fitting" would be better. This second sort of matching usually has a top-down or expectation-driven flavor; a representation is being related to a preexisting one.

As a final extension to the meaning of matching, matching might include the process of checking a structure with a set of rules describing structural legality, consistency, or likelihood. In this sense a scene can be matched against rules to see if it is nonsense or to assign an interpretation. One such interpretation process (called *labeling*) assigns consistent or optimally likely interpretations (labels) at one level to entities of another level. Labeling is like matching a given structure with a possibly infinite set of acceptable structures to find the best fit. However, we (fairly arbitrarily) treat labeling in Chapter 12 as extended inference rather than here as extended matching.

11.1.2 Matching Iconic, Geometric, and Relational Structures

Chapter 3 presented various correlation techniques for matching *iconic* (image-like) structures with each other. The bulk of this chapter, starting in Section 11.2, deals with matching *relational* (semantic net) structures. Another important sort of matching between two dissimilar representations fits data to parameterized models (usually geometric). This kind of matching is an important part of computer vi-

sion. A typical example is shown in Fig. 11.2. A preexisting representation (here a straight line) is to be used to interpret a set of input data. The line that best "explains" the data is (by definition) the line of "best fit." Notice that the decision to use a line (rather than a cubic, or a piecewise linear template) is made at a higher level. Given the model, the fitting or matching means determining the *parameters* of the model that tailor it into a useful abstraction of the data.

Sometimes there is no parameterized mathematical model to fit, but rather a given geometric structure, such as a piecewise linear curve representing a shoreline in a map which is to be matched to a piece of shoreline in an image, or to another piecewise linear structure derived from such a shoreline. These geometric matching problems are not traditional mathematical applications, but they are similar in that the best match is defined as the one minimizing a measure of disagreement.

Often, the computational solutions to such geometric matching problems exhibit considerable ingenuity. For example, the shore-matching example above may proceed by finding that position for the segment of shore to be matched that minimizes some function (perhaps the square) of a distance metric (perhaps Euclidean) between input points on the iconic image shoreline and the nearest point on the reference geometric map shoreline. To compute the smallest distance between an arbitrary point and a piecewise linear point set is not a trivial task, and this calculation may have to be performed often to find the best match. The computation may be reduced to a simple table lookup by precomputing the metric in a "chamfer array," that contains the metric of disagreement for any point around the geometric reference shoreline [Barrow et al. 1978]. The array may be computed efficiently by symmetric axis transform techniques (Chapter 8) that "grow" the linear structure outward in contours of equal disagreement (distance) until a value has been computed for each point of the chamfer array.

Parameter optimization techniques can relate geometrical structures to lower-level representations and to each other through the use of a merit function measuring how well the relations match. The models are described by a vector of parameters $\mathbf{a} = (a_1,\ldots,a_n)$. The merit function M must rate each set of those parameters in terms of a real number. For example, M could be a function of both a, the parameters, and $f(x)$, the image. The problem is to find a such that

$$M(\mathbf{a}, f(\mathbf{x}))$$

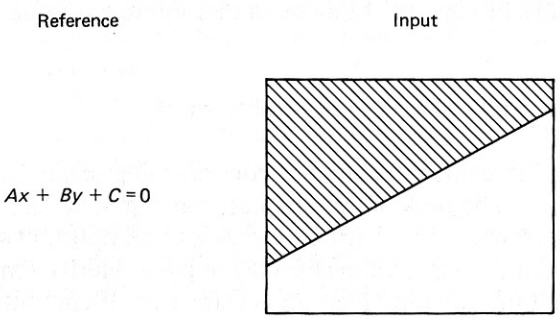


Fig. 11.2 Matching or fitting a straight line model to data.

is maximized. Note that if **a** were some form of template function rather than a vector of parameters, the problem statement would encompass the iconic correlation techniques just covered. There is a vast literature on optimization techniques and we cannot do more than provide a cursory discussion of a few cases with examples.

Formally, the different techniques have to do with the form of the merit function M. A fundamental result from calculus is that if M is sufficiently well behaved (i.e., has continuous derivatives), then a condition for a local maximum (or minimum) is that

$$M_{a_j} = \frac{\partial M}{\partial a_j} = 0 \qquad \text{for } j = 1,\ldots,n \tag{11.1}$$

This condition can be exploited in many different ways.

- Sometimes Eqs. (11.1) are sufficiently simple so that the a can be determined analytically, as in the least squares fitting, described in Appendix 1.
- An approximate solution a^0 can be iteratively adjusted by moving in the gradient direction or direction of maximum improvement:

$$a_j^k = a_j^{k-1} + cM_{a_j} \tag{11.2}$$

  where c is a constant. This is the most elementary of several kinds of *gradient (hill-climbing) techniques.* Here the gradient is defined with respect to M and does not mean edge strength.
- If the partial derivatives are expensive to calculate, the coefficients can be perturbed (either randomly or in a structured way) and the perturbations kept if they improve M:

  (1) $\mathbf{a}' := \mathbf{a} + \Delta\mathbf{a}$

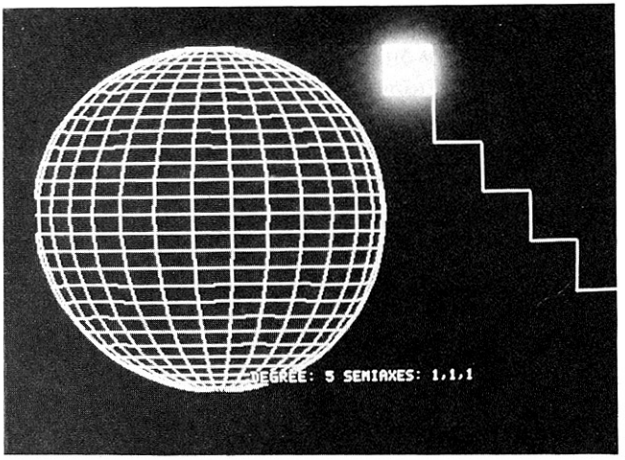
  (2) $\mathbf{a} := \mathbf{a}'$ if $M(\mathbf{a}') > M(\mathbf{a})$

A program to fit three-dimensional image data with shapes described by spherical harmonics used these techniques [Schudy and Ballard 1978]. The details of the spherical harmonics shape representation appear in Chapter 9. The fitting proceeded by the third method above. A nominal expected shape was matched to boundaries in image data. If a subsequent perturbation in one of its parameters results in an improvement in fit it was kept; otherwise, a different perturbation was made. Figure 11.3 shows this fitting process for a cross section of the shape.

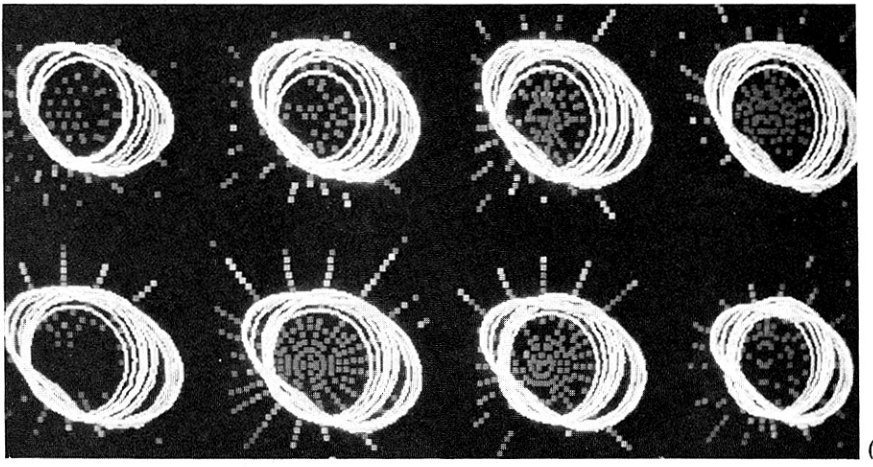
Though parameter optimization is an important aspect of matching, we shall not pursue it further here in view of the extensive literature on the subject.

11.2 GRAPH-THEORETIC ALGORITHMS

The remainder of this chapter deals with methods of matching relational structures. Chapter 10 showed how to represent a relational structure containing n-ary relations as a graph with labeled arcs. Recall that the labels can have values from a



(a)



(b)

(c)

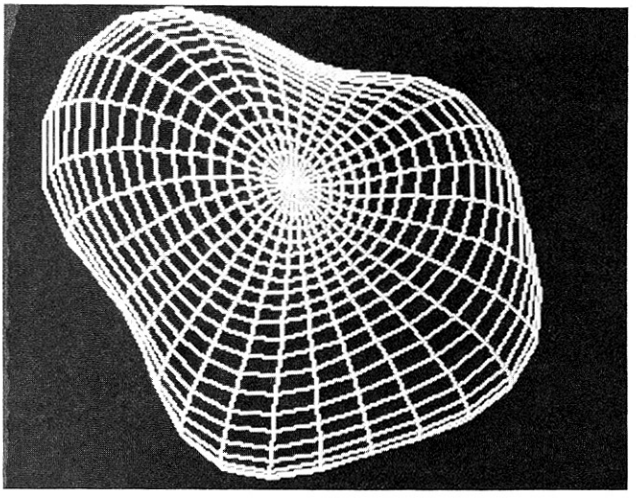


Fig. 11.3 An example of matching as parameter optimization. (a) Initial parameter set (displayed at left as three-dimensional surface (see Fig. 9.8) (b) Fitting process: iteratively adjust a based on M (see text). (c) Final parameter set yields this three-dimensional surface. (*See color inserts.*)

continuum, and that labeled arcs could be replaced by nodes to yield a directed graph with labeled nodes.

Depending on the attributes of the relational structure and of the correspondence desired, the definition of a match may be more or less elegant. It is always possible to translate powerful representations such as labeled graphs or n-ary relations into computational representations which are amenable to formal treatment (such as undirected graphs). However, when graph algorithms are to be implemented with computer data structures, the freedom and power of programming languages often tempts the implementer away from pure graph theory. He can replace elegant (but occasionally restrictive and impractical) graph-theoretic concepts and operations with arbitrarily complex data structures and algorithms.

One example is the "graph isomorphism" problem, a very pure version of relational structure matching. In it, all graph nodes and arcs are unlabeled, and graphs match if there is a 1:1 and onto correspondence between the arcs and nodes of the two graphs. The lack of expressive power in these graphs and the requirement that a match be "perfect" limits the usefulness of this pure model of matching in the context of noisy input and imprecise reference structures. In practice, graph nodes may have properties with continuous ranges of values, and an arbitrarily complex algorithm determines whether nodes or arcs match. The algorithm may even access information outside the graphs themselves, as long as it returns the answer "match" or "no match." Generalizing the graph-theoretic notions in this way can obscure issues of their efficiency, power, and properties; one must steer a course between the "elegant and unusable" and the "general and uncontrollable." This section introduces some "pure" graph-theoretic algorithms that form the basis for techniques in Sections 11.3 and 11.4.

11.2.1 The Algorithms

The following are several definitions of matching between graphs [Harary 1969; Berge 1976].

- *Graph isomorphism.* Given two graphs (V_1, E_1) and (V_2, E_2), find a 1:1 and onto mapping (an isomorphism) f between V_1 and V_2 such that for $v_1, v_2 \in V_1, V_2$, $f(v_1) = v_2$ and for each edge of E_1 connecting any pair of nodes v_1 and $v'_1 \in V_1$, there is an edge of E_2 connecting $f(v_1)$ and $f(v_1')$.
- *Subgraph isomorphism.* Find isomorphisms between a graph (V_1, E_1) and *subgraphs* of another graph (V_2, E_2). This is computationally harder than isomorphism because one does not know in advance which subsets of V_2 are involved in isomorphisms.
- *"Double" subgraph isomorphisms.* Find all isomorphisms between *subgraphs* of a graph (V_1, E_1) and *subgraphs* of another graph (V_2, E_2). This sounds harder than the subgraph isomorphism problem, but is equivalent.
- A match may not conform to strict rules of correspondence between arcs and nodes (some nodes and arcs may be "unimportant"). Such a matching criterion may well be implemented as a "computational" (impure) version of one of the pure graph isomorphisms.

Figure 11.4 shows examples of these kinds of matches.

One algorithm for finding graph isomorphism [Corneil and Gotlieb 1970] is based on the idea of separately putting each graph into a canonical form, from which isomorphism may easily be determined. For directed graphs (i.e., nonsymmetric relations) a backtrack search algorithm [Berztiss 1973] works on both graphs at once.

Two solutions to the subgraph isomorphism problem appear in [Ullman 1976]: The first is a simple enumerative search of the tree of possible matches between nodes. The second is more interesting; in it a process of "parallel-iterative" refinement is applied at each stage of the search. This process is a way of rejecting node pairs from the isomorphism and of propagating the effects of such rejections; one rejected match can lead to more matches being rejected. When the iteration converges (i.e., when no more matches can be rejected at the current stage), another step in the tree search is performed (one more matching pair is hypothesized). This mixing of parallel-iterative processes with tree search is useful in a variety of applications (Section 11.4.4, Chapter 12).

"Double" subgraph isomorphism is easily reduced to subgraph isomorphism via another well-known graph problem, the "clique problem." A *clique* of size N is a totally connected subgraph of size N (each node is connected to every other node in the clique by an arc). Finding isomorphisms between subgraphs of a graph A and subgraphs of a graph B is accomplished by forming an *association graph* G from the graphs A and B and finding cliques in G (for details, see Section 11.3.3). Clique

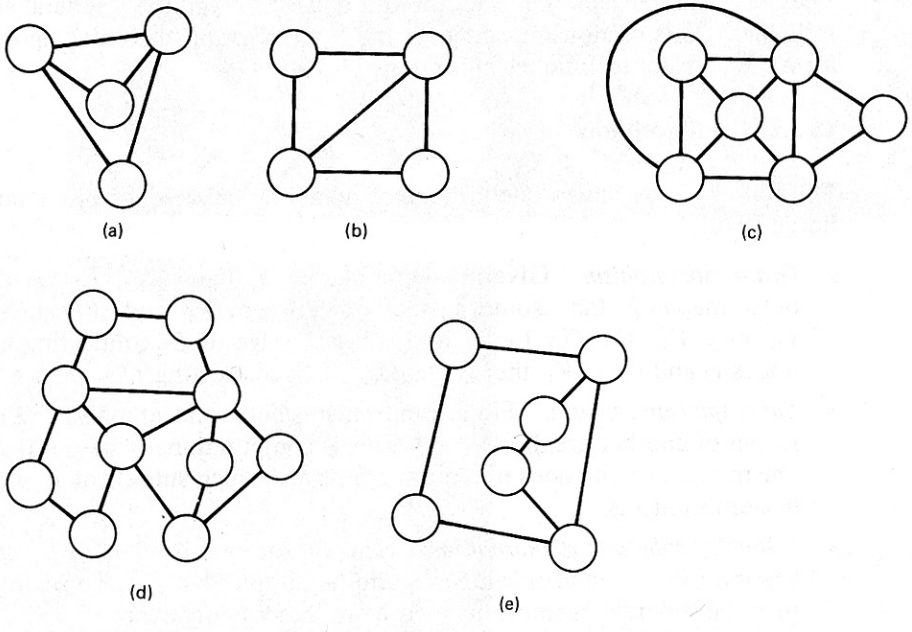


Fig. 11.4 Isomorphisms and matches. The graph (a) has an isomorphism with (b), various subgraph isomorphisms with (c), and several "double" subgraph isomorphisms with (d). Several partial matches with (e) (and also (b), (c), and (d)), depending on which missing or extra nodes are ignored.

finding may be done with a subgraph isomorphism algorithm; hence the reduction. Several other clique-finding algorithms exist [Ambler et al. 1975; Knodel 1968; Bron and Kerbosch 1973; Osteen and Tou 1973].

11.2.2 Complexity

It is of some practical importance to be aware of the computational complexity of the matching algorithms proposed here; they may take surprising amounts of computer time. There are many accessible treatments of computational complexity of graph-theoretic algorithms [Reingold et al. 1977; Aho, Hopcroft and Ullman 1974]. Theoretical results usually describe worst-case or average time complexity. The state of knowledge in graph algorithms is still improving; some interesting worst-case bounds have not been established.

A "hard" combinatorial problem is one that takes time (in a usual model of computation based on a serial computer) proportional to an exponential function of the length of the input. "Polynomial-time" solutions are desirable because they do not grow as fast with the size of the problem. The time to find all the cliques of a graph is in the worst case inherently exponential in the size of the input graphs, because the output is an exponential number of graphs. Both the single subgraph isomorphism problem and the "clique problem" (does there exist a clique of size k?) are *NP-complete*; all known deterministic algorithms run (in the worst case) in time exponential in the length of the description of the graphs involved (which must specify the nodes and arcs). Not only this, but if either of these problems (or a host of other NP complete problems) could be solved deterministically in time polynomially related to the length of the input, it could be used to solve all the other NP problems in polynomial time.

Graph isomorphism, both directed and undirected, is at this writing in a netherworld (along with many other combinatorial problems). No polynomial-time deterministic algorithms are known to exist, but the relation of these problems to each other is not as clear-cut as it is between the NP-complete problem. In particular, finding a polynomial-time deterministic solution to one of them would not necessarily indicate anything about how to solve the other problems deterministically in polynomial time. These problems are not mutually reducible. Certain restrictions on the graphs, for instance that they are planar (can be arranged with their nodes in a plane and with no arcs crossing), can make graph isomorphism an "easy" (polynomial-time) problem.

The average-case complexity is often of more practical interest than the worst case. Typically, such a measure is impossible to determine analytically and must be approximated through simulation. For instance, one algorithm to find isomorphisms of randomly generated graphs yields an average time that seems not exponential, but proportional to N^3, with N the number of nodes in the graph [Ullman 1976]. Another algorithm seems to run in average time proportional to N^2 [Corneil and Gotlieb 1970].

All the graph problems of this section are in NP. That is, a *non*deterministic algorithm can solve them in polynomial time. There are various ways of visualizing

nondeterministic algorithms; one is that the algorithm makes certain significant "good guesses" from a range of possibilities (such as correctly guessing which subset of nodes from graph B are isomorphic with graph A and then only having to worry about the arcs). Another way is to imagine *parallel* computation; in the clique problem, for example, imagine multiple machines running in parallel, each with a different subset of nodes from the input graph. If any machine discovers a totally connected subset, it has, of course, discovered a clique. Checking whether N nodes are all pairwise connected is at most a polynomial-time problem, so all the machines will terminate in polynomial time, either with success or not. Several interesting processes can be implemented with parallel computations. Ullman's algorithm uses a refinement procedure which may run in parallel between stages of his tree search, and which he explains how to implement in parallel hardware [Ullman 1976].

11.3 IMPLEMENTING GRAPH-THEORETIC ALGORITHMS

11.3.1 Matching Metrics

Matching involves *quantifiable similarities.* A match is not merely a correspondence, but a correspondence that has been quantified according to its "goodness." This measure of goodness is the *matching metric.* Similarity measures for correlation matching are lumped together as one number. In relational matching they must take into account a relational, structured form of data [Shapiro and Haralick 1979].

Most of the structural matching metrics may be explained with the physical analogy of "templates and springs" [Fischler and Elschlager 1973]. Imagine that the reference data comprise a structure on a transparent rubber sheet. The matching process moves this sheet over the input data structure, distorting the sheet so as to get the best match. The final goodness of fit depends on the individual matches between elements of the input and reference data, and on the amount of work it takes to distort the sheet. The continuous deformation process is a pretty abstraction which most matching algorithms do not implement. A computationally more tractable form of the idea is to consider the model as a set of rigid "templates" connected by "springs" (see Fig. 11.5). The templates are connected by "springs" whose "tension" is also a function of the relations between elements. A spring function can be arbitrarily complex and nonlinear; for example the "tension" in the spring can attain very high or infinite values for configurations of templates which cannot be allowed. Nonlinearity is good for such constraints as: in a picture of a face the two eyes must be essentially in a horizontal line and must be within fixed limits of distance. The quality of the match is a function of the goodness of fit of the templates locally and the amount of "energy" needed to stretch the springs to force the input onto the reference data. Costs may be imposed for missing or extra elements.

The template match functions and spring functions are general procedures, thus the templates may be more general than pure iconic templates. Further,