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,

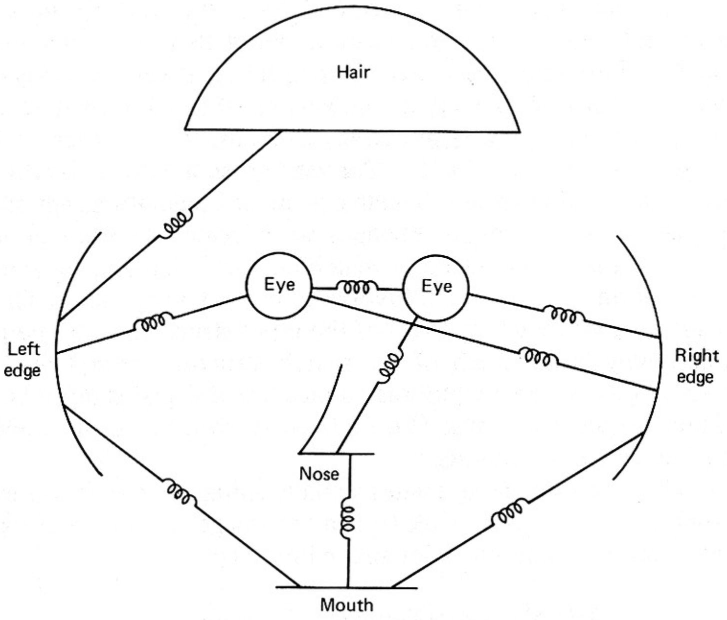

Fig. 11.5 A templates and springs model of a face.

matches may be defined not only between nodes and other nodes, but between nodes and image data directly. Thus the template and springs formalism is workable for "cross-representational" matching. The mechanism of minimizing the total cost of the match can take several forms; more detailed examples follow in Section 11.4.

Equation 11.3 a general form of the template-and-springs metric. TemplateCost measures dissimilarity between the input and the templates, and SpringCost measures the dissimilarity between the matched input elements' relations and the reference relations between the templates. MissingCost measures the penalties for missing elements. F(·) is the mapping from templates of the reference to elements of the input data. F partitions the reference templates into two classes, those found {FoundinRefer} and those not found {MissinginRefer} in the input data. If the input data are symbolic they may be similarly partitioned. The general metric is

$$\begin{aligned}\text{Cost} = &\sum_{d \,\in\, \{\text{FoundinRefer}\}} \text{TemplateCost}(d, F(d)) \\ &+ \sum_{(d,\, e) \,\in\, \{\text{FoundinRefer} \times \text{FoundinInput}\}} \text{SpringCost}(F(d), F(e)) \\ &+ \sum_{c \,\in\, \{\text{MissinginRefer}\} \,\bigcup\, \{\text{MissinginInput}\}} \text{MissingCost}(c)\end{aligned} \qquad (11.3)$$

Equation 11.3 may be written as one sum of generalized SpringCosts in which the template properties are included (as 1-ary relations), as are "springs" involving missing elements.

As with correlation metrics, there are normalization issues involved with structural matching metrics. The number of elements matched may affect the ultimate magnitude of the metric. For instance, if springs always have a finite cost, then the more elements that are matched, the higher the total spring energy must be; this should probably not be taken to imply that a match of many elements is worse than a match of a few. Conversely, suppose that relations which agree are given positive "goodness" measures, and a match is chosen on the basis of the total "goodness." Then unless one is careful, the sheer number of possibly mediocre relational matches induced by matching many elements may outweigh the "goodness" of an elegant match involving only a few elements. On the other hand, a small, elegant match of a part of the input structure with one particular reference object may leave much of the search structure unexplained. This good "submatch" may be less helpful than a match that explains more of the input. To some extent the general metric (Eq.11.3) copes with this by acknowledging the "missing" category of elements.

If the reference templates actually contain iconic representations of what the input elements should look like in the image, a TemplateCost can be associated with a template and a location in the image by

$$\begin{aligned}&\text{TemplateCost(Template, Location)}\\&\qquad = (1 - \text{normalized correlation metric between}\\&\qquad\quad \text{template shape and input image at the location}).\end{aligned}$$

If the match is, for instance, to match reference descriptions of a chair with an input data structure, a typical "spring" might be that the chair seat must be supported by its legs. Thus if F is the association function mapping reference elements such as LEG or TABLETOP to input elements,

$$\text{SpringCost}_1(F(\text{LEG}), F(\text{TABLETOP})) = \begin{cases} 0 & \text{if } F(\text{LEG}) \text{ appears to support } F(\text{TABLETOP}), \\ 1 & \text{if } F(\text{LEG}) \text{ does not appear to support } F(\text{TABLETOP}). \end{cases}$$

For quantified relations, one might have

$$\text{SpringCost}_2 = \text{number of standard deviations from the canonical mean value for this relation.}$$

Another version of SpringCost_2 is the following [Barrow and Popplestone 1971].

$$\text{Cost of Match} = \frac{\text{SpringCosts of properties (unary) and binary relations}}{\text{total number of unary and binary springs}} + \frac{\text{Empirical Constant}}{\text{Total number of reference elements matched}} \qquad (11.4)$$

The first term measures the average badness of matches between properties (unary relations) and relations between regions. The second term is inversely proportional to the number of regions that are matched, effectively increasing the cost of matches that explain less of the input.

11.3.2 Backtrack Search

Backtrack search is a generic name for a type of potentially exhaustive search organized in stages; each processing stage attempts to extend a partial solution derived in the previous stage. Should the attempt fail, the search "backtracks" to the most recent partial solution, from which a new extension is attempted. The technique is basic, amounting to a depth-first search through a tree of partial solutions (Fig. 11.6). Backtracking is a pervasive control structure in artificial intelli-

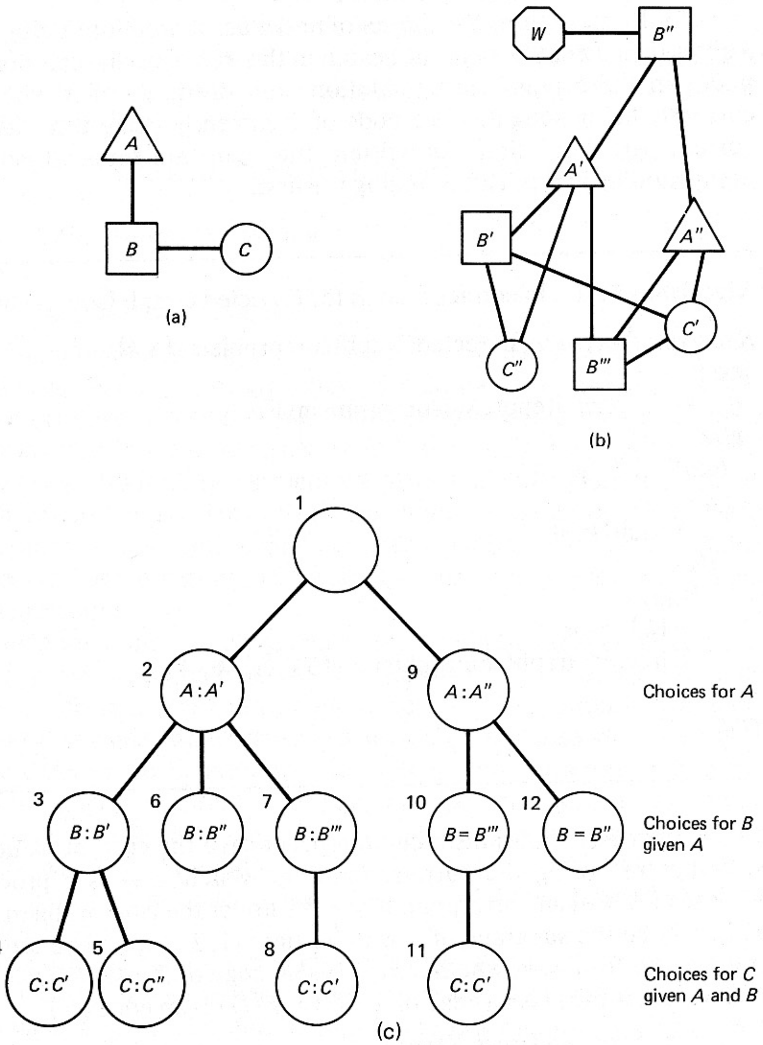

Fig. 11.6 The graph of (a) is to be matched in (b) with arcs all being unlabeled but nodes having properties indicated by their shapes, (c) is the tree of solutions built by a backtrack algorithm.

gence, and through the years several general classes of techniques have evolved to make the basic, brute-force backtrack search more efficient.

Example: Graph Isomorphisms

Given two graphs,

$$X = (V_X, E_X)$$
$$Y = (V_Y, E_Y),$$

without loss of generality, let $V_X = V_Y = \{1, 2, \ldots, n\}$, and let X be the reference graph, Y the input graph. The isomorphism is given by: If $i \in V_X$, the corresponding node under the isomorphism is $F(i) \in V_Y$.

In the algorithm, S is the set of nodes accounted for in Y by a partial solution. k gives the current level of the search in the tree of partial solutions, the number of nodes in the current partial solution, and the node of X whose match in Y is currently being sought. v is a node of Y currently being considered to extend the current partial solution. As written, the algorithm finds all isomorphisms. It is easily modified to quit after finding the first.

Algorithm 11.1 Backtrack Search for Directed Graph Isomorphism

```
Recursive Procedure DirectedGraphIsomorphisms($S,k$);
begin
 if $S=V_Y$ then ReportAsIsomorphism(F)
 else
  forall $v \in (V_Y-S)$
   do
   if Match($k,v$)
   then
    begin
     $F(k) := v$;
     DirectedGraphIsomorphisms ($S \in \{v\}, k+1$);
    end;
end;
```

ReportAsIsomorphism could print or save the current value of F, the global structure recording the current solution. Match(k,v) is a procedure that tests whether $v \in V_Y$ can correspond to $k \in V_X$ under the isomorphism so far defined by F. Let X_k be the subgraph of X with vertices $\{1,2,\ldots,k\}$. The procedure "Match" must check for $i < k$, whether (i, k) is an edge of X_k iff $(F(i), v)$ is an edge of Y *and* whether (k, i) is an edge of X_k iff $(v, F(i))$ is an edge of Y.

Improving Backtrack Search

Several techniques are useful in improving the efficiency of backtrack search [Bittner and Reingold 1975]:

1. *Branch pruning.* All techniques of this variety examine the current partial solution and prune away descendents that are not viable continuations of the solution. Should none exist, backtracking can take place immediately.
2. *Branch merging.* Do not search branches of the solution tree isomorphic with those already searched.
3. *Tree rearrangement and reordering.* Given pruning capabilities, more nodes are likely to be eliminated by pruning if there are fewer choices to make early in the search (partial solution nodes of low degree should be high in the search tree). Similarly, search first those extensions to the current solution that have the fewest alternatives.
4. *Branch and bound.* If a cost may be assigned to solutions, standard techniques such as heuristic search and the A* search algorithm [Nilsson 1980] (Section 4.4) may be employed to allow the search to proceed on a "best-first" rather than a "depth-first" basis.

For extensions of these techniques, see [Haralick and Elliott 1979].

11.3.3 Association Graph Techniques

Generalized Structure Matching

A general relational structure "best match" is less restricted than graph isomorphism, because nodes or arcs may be missing from one or the other graph. Also, it is more general than subgraph isomorphism because one structure may not be exactly isomorphic to a substructure of the other. A more general match consists of a set of nodes from one structure and a set of nodes from the other and a 1:1 mapping between them which preserves the compatibilities of properties and relations. In other words, corresponding nodes (under the node mapping) have sufficiently similar properties, and corresponding sets under the mapping have compatible relations.

The two relational structures may have a complex makeup that falls outside the normal purview of graph theory. For instance, they may have parameterized properties attached to their nodes and edges. The definition of whether a node matches another node and whether two such node matches are mutually compatible can be determined by arbitrary procedures, unlike the much simpler criteria used in pure graph isomorphism or subgraph isomorphism, for example. Recall that the various graph and subgraph isomorphisms rely heavily on a 1:1 match, at least locally, between arcs and nodes of the structures to be matched. However, the idea of a "best match" may make sense even in the absence of such perfect correspondences.

The *association graph* defined in this section is an auxiliary data structure produced from two relational structures to be matched. The beauty of the association graph is that it *is* a simple, pure graph-theoretic structure which is amenable to pure graph-theoretic algorithms such as clique finding. This is useful for several reasons.

- It takes relational structure matching from the ad hoc to the classical domain.
- It broadens the base of people who are producing useful algorithms for structure matching. If the rather specialized relational structure matching enterprise is reducible to a classical graph-theoretical problem, then everyone working on the classical problem is also working indirectly on structure matching.
- Knowledge about the computational complexity of classical graph algorithms illuminates the difficulty of structure matching.

Clique Finding for Generalized Matching

Let a relational structure be a set of elements V, a set of properties (or more simply unary predicates) P defined over the elements, and a set of binary relations (or binary predicates) R defined over pairs of the elements. An example of a graph representation of such a structure is given in Fig. 11.7.

Given two structures defined by (V_1, P, R) and (V_2, P, R), say that "similar" and "compatible" actually mean "the same." Then we construct an association graph G as follows [Ambler et al. 1975]. For each v_1 in V_1 and v_2 in V_2, construct a node of G labeled (v_1, v_2) if v_1 and v_2 have the same properties [$p(v_1)$ iff $p(v_2)$ for each p in P]. Thus the nodes of G denote assignments, or pairs of nodes, one each from V_1 and V_2, which have similar properties. Now connect two nodes (v_1, v_2) and (v'_1, v'_2) of G if they represent *compatible* assignments according to R, that is, if the pairs satisfy the same binary predicates [$r(v_1, v'_1)$ iff $r(v_2, v'_2)$ for each r in R].

A match between (V_1, P, R) and (V_2, P, R), the two relational structures, is just a set of assignments that are all mutually compatible. The "best match" could well be taken to be the largest set of assignments (node correspondences) that were all mutually compatible under the relations. But this in the association graph G is just the largest totally connected (completely mutually compatible)—set of nodes. It is a *clique.* A clique to which no new nodes may be added without destroying the clique properties is a *maximal* clique. In this formulation of matching, larger cliques are taken to indicate better matches, since they account for more nodes.

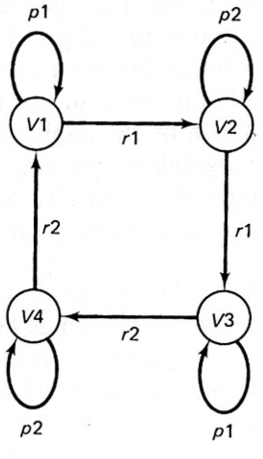

Fig. 11.7 A graph representation of a relational structure. Elements (nodes) v_1 and v_3 have property p1, v_2 and v_4 have property p2, and the arcs between nodes indicate that the relation r1 holds between v_1 and v_2 and between v_2 and v_3, and r2 holds between v_3 and v_4 and between v_4 and v_1.

Thus the best matches are determined by the largest maximal cliques in the association graph. Figure 11.8 shows an example: Certain subfeatures of the objects have been selected as "primitive elements" of the objects, and appear as nodes (elements) in the relational structures. To these nodes are attached properties, and between them can exist relations. The choice of primitives, properties, and relations is up to the designer of the representation. Here the primitives of the representation correspond to edges and corners of the shape.

The association graph is shown in 11.8e. Its nodes correspond to pairs of nodes, one each from A and B, whose properties are similar. [Notice that there is no node in the association graph for (6,6′)]. The arcs of the association graph indicate that the endpoints of the arc represent compatible associations. Maximal cliques in the association graph (shown as sets of nodes with the same shape) indicate sets of consistent associations. The largest maximal clique provides the node pairings of the "best match."

In the example construction, the association graph is formed by associating nodes with exactly the same properties (actually unary predicates), and by allowing as compatible associations only those with exactly the same relations (actually binary predicates). These conditions are easy to state, but they may not be exactly what is needed. In particular, if the properties and relations may take on ranges of values greater than the binary "exists" and "does not exist," then a measure of similarity must be introduced to define when node properties are similar enough for association, and when relations are similar enough for compatibility. Arbitrarily complex functions can decide whether properties and relations are similar. As long as the function answers "yes" or "no," the complexity of its computations is irrelevant to the matching algorithm.

The following recursive clique-finding algorithm builds up cliques a node at a time [Ambler et al. 1975]. The search tree it generates has states that are ordered pairs (set of nodes chosen for a clique, set of nodes available for inclusion in the clique). The root of the tree is the state (∅, all graph nodes), and at each branch a choice is made whether to include or not to include an eligible node in the clique. (If a node is eligible for inclusion in clique X, then *each* clique including X must either include the node or exclude it).

Algorithm 11.2: Clique-Finding Algorithm

Comment *Nodes* is the set of nodes in the input graph.

```
Comment
  Cliques (X, Y)  takes as arguments a clique X, and Y, a set of nodes that includes
                  X. It returns all cliques that include X and are included in Y.
  Cliques (∅,Nodes) finds all cliques in the graph.
Cliques(X, Y) :=
  if no node in Y−X is connected to all elements of X
    then {X}
    else
      Cliques(X ∪ {y}, Y) ∪ Cliques (X, Y−{y})
      where y is connected to all elements of X.
```

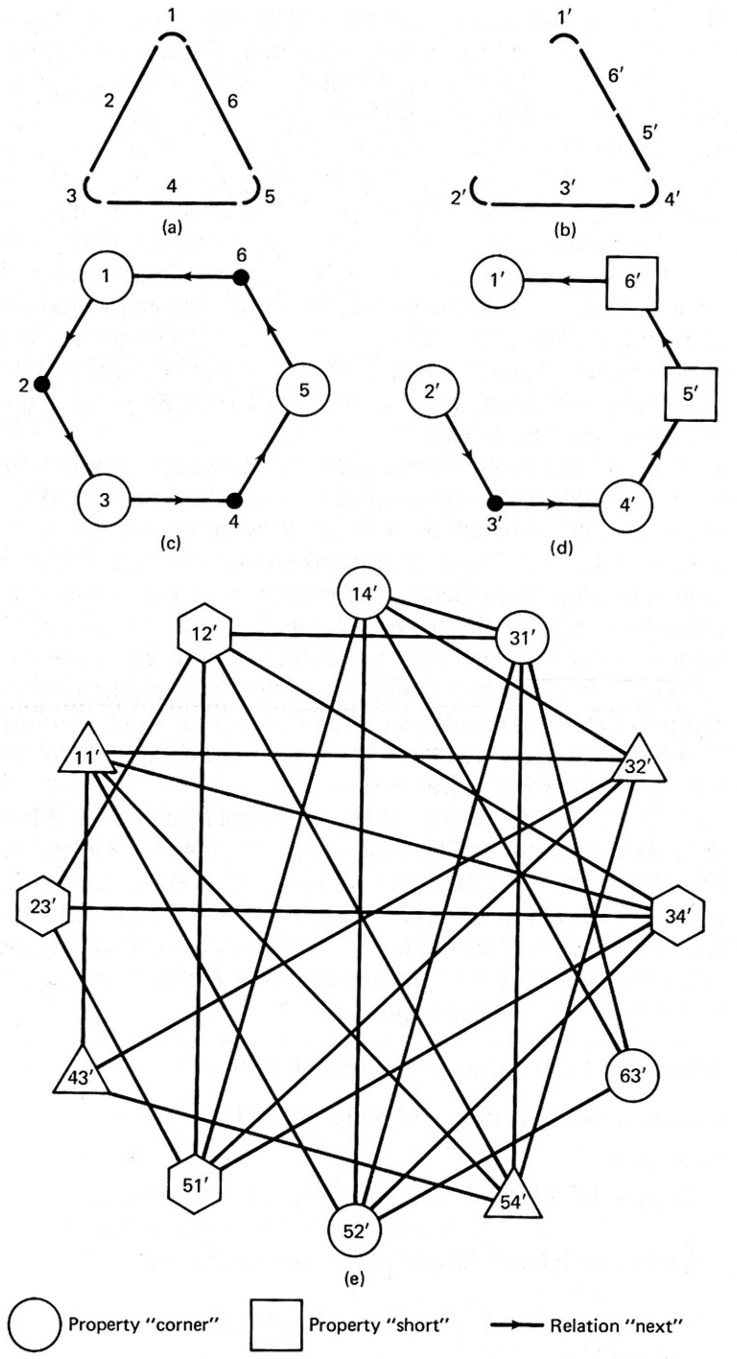

Fig. 11.8 Clique-finding example. Entities to be matched are given in (a) (reference) and (b) (input). The relational structures corresponding to them are shown in (c) and (d). The resulting association graph is shown in (e) with its largest cliques indicated by node shapes.

Extensions

Modifications to the clique-finding algorithm extend it to finding maximal cliques and finding largest cliques. To find largest cliques, perform an additional test to stop the recursion in *Cliques* if the size of X plus the number of nodes in $Y-X$ connected to all of X becomes less than k, which is initially set to the size of the largest possible clique. If no cliques of size k are found, decrement k and run *Cliques* with the new k.

To find maximal cliques, at each stage of *Cliques*, compute the set

$$Y' = \{z \in \text{Nodes: } z \text{ is connected to each node of Y}\}.$$

Since any maximal clique must include Y', searching a branch may be terminated should Y' not be contained in Y, since Y can then contain no maximal cliques.

The association graph may be searched not for cliques, but for r-connected components. An *r-connected* component is a set of nodes such that each node is connected to at least r other nodes of the set. A clique of size n is an $n-1$-connected component. Fig. 11.9 shows some examples.

The r-connected components generalize the notion of clique. An r-connected component of N nodes in the association graph indicates a match of N pairs of nodes from the input and reference structures, as does an N-clique. Each matching pair has similar properties, and each pair is compatible with at least r other matches in the component.

Whether or not the r-connected component definition of a match between two structures is useful depends on the semantics of "compatibility." For instance, if all relations are either compulsory or prohibited, clearly a clique is called for. If the relations merely give some degree of mutual support, perhaps an r-connected component is the better definition of a match.

11.4 MATCHING IN PRACTICE

This section illustrates some principles of matching with examples from the computer vision literature.

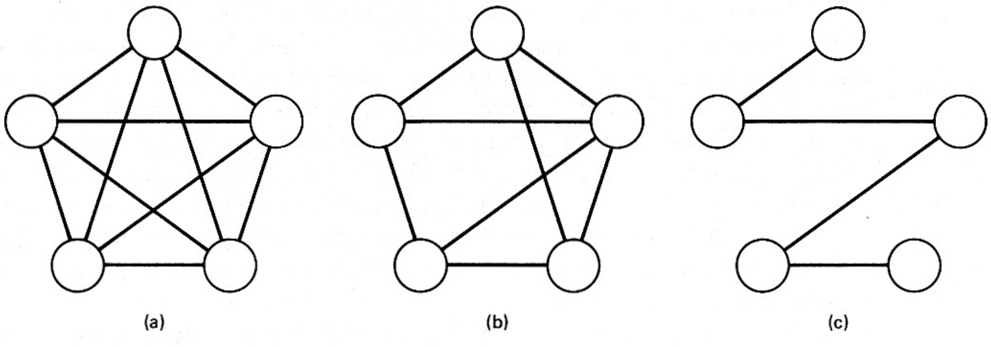

Fig. 11.9 r-connected components. (a) A 5-clique (which is 4-connected). (b) A 3-connected set of 5 nodes. (c) A 1-connected set of 5 nodes.