Error Checking and Graphical Representation of Multiple–Complete –Digest (MCD) Restriction-Fragment Maps

Edward C. Thayer, Maynard V. Olson and Richard M. Karp

Genome Res. 1999 9: 79-90
Access the most recent version at doi:10.1101/gr.9.1.79

Error Checking and Graphical Representation of Multiple–Complete–Digest (MCD) Restriction-Fragment Maps

Edward C. Thayer,[1] Maynard V. Olson, and Richard M. Karp[2]

University of Washington Genome Center, Seattle, Washington 98195 USA; [2]Department of Computer Science and Engineering, University of Washington, Seattle, Washington 98195 USA

Genetic and physical maps display the relative positions of objects or markers occurring within a target DNA molecule. In constructing maps, the primary objective is to determine the ordering of these objects. A further objective is to assign a coordinate to each object, indicating its distance from a reference end of the target molecule. This paper describes a computational method and a body of software for assigning coordinates to map objects, given a solution or partial solution to the ordering problem. We describe our method in the context of multiple–complete–digest (MCD) mapping, but it should be applicable to a variety of other mapping problems. Because of errors in the data or insufficient clone coverage to uniquely identify the true ordering of the map objects, a partial ordering is typically the best one can hope for. Once a partial ordering has been established, one often seeks to overlay a metric along the map to assess the distances between the map objects. This problem often proves intractable because of data errors such as erroneous local length measurements (e.g., large clone lengths on low-resolution physical maps). We present a solution to the coordinate assignment problem for MCD restriction-fragment mapping, in which a coordinated set of single-enzyme restriction maps are simultaneously constructed. We show that the coordinate assignment problem can be expressed as the solution of a system of linear constraints. If the linear system is free of inconsistencies, it can be solved using the standard Bellman–Ford algorithm. In the more typical case where the system is inconsistent, our program perturbs it to find a new consistent system of linear constraints, close to those of the given inconsistent system, using a modified Bellman–Ford algorithm. Examples are provided of simple map inconsistencies and the methods by which our program detects candidate data errors and directs the user to potential suspect regions of the map.

"Marker-content" methods play a major role in genome mapping. A simple example is STS-content mapping (Olson and Green 1990), which is based on measurements of incidences between a set of clones and a set of STSs (sequence-tagged sites) sampled from the region that is to be mapped. Radiation-hybrid mapping and genetic mapping on reference pedigrees also follow the marker-content paradigm. In all cases, subintervals of the genome are defined in some way: as recombinant-DNA molecules, fragments propagated in somatic cell hybrid lines, or regions between meiotic crossovers. These subintervals are tested, either individually or in pools, for the presence of STSs or genetic markers that distinguish between different haplotypes.

It is a characteristic of these methods that the data underdetermine the coordinates of the markers and the interval end points (Green and Green 1991; Olson and Green 1993). Even in the case of idealized, error-free data on "mapping panels" that provide many interval ends per marker, these methods determine the order of the markers, whereas they only set bounds on the coordinates of both the markers and the interval end points. As the number of interval ends per marker drops, even marker ordering is incompletely determined (Olson and Green 1993). In the presence of inevitable data errors, marker-content methods still suffer from global underdetermination of coordinates, and they also display local instances of overdetermination, in which there is no set of marker and interval-end coordinates that is fully consistent with the data.

We describe here a computational approach to a special case of this problem that arises in multiple–complete–digest (MCD) mapping, a restricton-fragment mapping method. MCD mapping is based on complete digestion of a redundant set of overlapping clones with each of several restriction enzymes (Gillett et al. 1996; Wong et al. 1997). MCD mapping can be thought of as a marker-content method in which the markers are the restriction fragments and the genomic subintervals are the ends of the recombinant-DNA molecules that were digested with restriction enzymes during the mapping process. MCD-mapping algorithms that lead to partial orderings of clone ends and restriction fragments have been implemented in the software package DNAM (Gillett et al. 1996). DNAM provides maps in which the partial orderings are guaranteed to be consistent with the underlying data. Hence, in the process of using DNAM to produce real maps, data errors that prevent construction of a self-

[1]Corresponding author. Present address: ZymoGenetics, Inc., Seattle, Washington 98102 USA.
E-MAIL thayer@zgi.com; FAX (206) 442-6608.

consistent ordering of clone ends and restriction fragments must be weeded out by the user. Moreover, DNAM takes no metric constraints into account; for example, it does not require that the maps constructed with the different restriction enzymes will be of the same length.

As a consequence of these properties, the output from DNAM displays the combination of under- and overdetermination of coordinates that is a hallmark of marker-content methods. Although the coordinates of many of the objects that one would like to place on an MCD map—particularly the clone ends—can only be specified within approximate bounds, there may exist no assignment of coordinates to all clone ends and restriction sites that is fully consistent with the data. Given the thousands or tens of thousands of inequalities that are required to describe all the geometric constraints that apply to a typical MCD map, any approach to developing a geometric interpretation of DNAM output must be computationally efficient. It should also provide as much useful information as possible to the experimentalists who are constructing and using MCD maps. We describe here a solution that meets these criteria for the specific case of MCD maps and illustrate principles that may also have wider application to other forms of mapping.

RESULTS AND DISCUSSION

DNAM Input, Fundamental Principles, and Output

To explain ATLAS in more detail, we must first explain the basic technique of MCD mapping (for a more detailed explanation of the algorithmic issues involved in MCD map assembly, see Gillett et al. 1996; for an alternative approach to MCD map construction, see Fasulo et al. 1996; Jiang and Karp 1998).

One starts the MCD mapping process by selecting a small number of restriction enzymes. Traditionally, the three enzymes *Eco*RI, *Hin*dIII, and *Nsi*I, which recognize 6-bp sites, have been used. Clones are selected from the target region, and each is completely digested by each of the enzymes. The fragments resulting from each complete digestion are sized by electrophoresis, and the collection of lists of fragment sizes is given to the mapping software DNAM. DNAM does not use the fragment length information to assess distances along the length of the map target. Rather, the fragment sizes are simply used to determine which fragments from different clones are eligible to be considered as representing the same fragment in the target DNA. The basic requirement is that if a set of fragments in the same enzyme domain is to be identified with a single fragment in the target DNA, the measured sizes of all the fragments in the set must agree within a specified tolerance. Except for this constraint, no use is made of the measured sizes of the fragments. In designing DNAM, the decision not to depend too heavily on the fragment length data was based on the uncertainty of fragment length measurements and their inability to detect small fragments that pass out the bottom of the gel during size separation. However, particularly when clones are collected at high redundancy, much length information would, in principle, be extractable from the data underlying the MCD maps.

The fragments within any clone A are initially unordered relative to one another, but the ordering relations among the clone ends induce ordering relations among the fragments. For example, if clones A and B intersect, then, in each enzyme domain, the fragments common to A and B must be consecutive in the ordering of A's fragments and in the ordering of B's. Similarly, if clone B lies strictly to the right of clone A, every fragment in clone B must lie to the right of every fragment in clone A. DNAM uses relations such as these among the clone ends to divide the fragments into *groups*, in which the ordering among the groups is strictly determined but the ordering within a group is unknown. Each group is bounded in the map by at least two specific clone ends. The success of the mapping process stems from the observation that the number of fragments in a group can get small when sufficiently many clones have been positioned and suitable identifications have been made between fragments within the different clones. Ideally, the fragments in each enzyme domain would be completely ordered, with each fragment assigned a unique position along the length of the target DNA. Full realization of this goal is impossible, especially when the clone coverage is low. Instead, one settles for a partition of the clones and fragments into contigs and, within each contig, a subdivision of the fragments into small groups and a complete ordering of the groups in each enzyme domain in relation to one another and to the clone ends. In short, DNAM creates a partial ordering of clone ends and fragments in each enzyme's map with the caveat that the clone-end ordering is consistent across enzymes. DNAM makes no attempt to estimate how far any two map objects are from each other. Such an estimation is essential to represent the map graphically.

Formalism for Estimating MCD Map Object Distances

There are many formalizations of the constraints between the various mapped objects given a partial ordering of clone ends and fragments. Fasulo et al. (1996) previously presented one formalization that arose from an alternative map assembly program than DNAM. We present a slightly different system of constraints. Both constraint systems can be solved using the modified Bellman–Ford algorithm described below.

Consider a contig with clone ends and fragment groups that have been partially ordered by DNAM. The map for each contig can be viewed as the result of

coordinating separate maps in different *enzyme domains*, where an enzyme domain contains all the measurements associated with a specific enzyme. Choosing either end of the contig as the zero position, denoted **O**, let l_i (or r_i) denote the distance from **O** to the nearest (or respectively, farthest) end of the *i*th clone. Once the contig end **O** has been chosen, the restriction sites forming the boundaries between adjacent groups of unordered fragments [referred to as *group boundaries* by Gillett et al. (1996)] can be ordered in accordance with their relative distances from **O**. We typically view the map with **O** at the leftmost position on the page, and therefore apply the term *leftmost* (or *rightmost*) to physical objects such as clone ends and restriction sites that occur closer (or respectively farther) from **O**. Given the ordering of the group boundaries across each enzyme domain, we let x_j^k represent the distance from **O** to the *j*th group boundary from the left of the *k*th enzyme domain. We adopt the notation convention that subscripts and superscripts for the variables l_i, r_i, and x_j^k will be dropped when it is clear from the context that a single, arbitrary clone or group is being referenced.

The linear nature of the modeled DNA segment imposes several constraints on the variables l_i, r_i, and x_j^k. Each of these constraints reflects a basic physical relationship of ''between-ness'' for two physical objects (clone ends and restriction fragments) and the fragments separating them on the map. Such a relationship translates to a linear inequality involving the two variables chosen to represent the positions of these physical objects. We then go through each constraint, giving both the linear inequality and an explanation of its origin in the MCD map.

Clone Length

Presented with an MCD map as constructed by DNAM, each map fragment in an enzyme domain has many representatives from different clones that have been identified as covering the same fragment of the mapping target. From this multiple sampling of the same fragment, we define the *virtual fragment length* as the average of these fragment lengths (as in Gillett et al. 1996). This undoubtedly helps correct for variations in the measurements of the fragment's length and therefore yields a more accurate estimate of each fragment's actual size.

For each clone in the map, the sum of the virtual lengths of the fragments completely contained in the clone in each enzyme domain varies enormously depending on the amount of unmapped DNA in the clone that results from partial fragments directly adjacent to the clone's vector and small fragments less than ~500 bp (see Fig. 1). But knowing the approximate length of the vector and using the hybridization data that identifies restriction fragments containing portions of the vector, we can estimate the length of the unmapped DNA in each enzyme domain and, in turn, arrive at an estimate for the total length of the clone's insert in each enzyme (see Fig. 1). These domain-specific clone insert lengths are then averaged across enzyme domains to produce an estimated global *clone length*, denoted by L. The length of the clone clearly constrains how far apart its left and right ends can be from each other, so that regardless of the clone's location along the map, l and r must be no further apart than L. Because L is an approximation of the clone's length, a strict equality relating l and r with L would be inappropriate. Hence, the clone length constraint is translated as two inequalities:

$$L - \alpha L \leq r - l \leq L + \alpha L \qquad (1)$$

in which α is a small positive number meant to ''soften'' the equality $r - l = L$. Generally, we take $\alpha = 0.001$. This reflects two mindsets about these numbers:

1. Experience has shown that our fragment sizing procedure is extremely accurate and that most errors are systematically dependent on fragment size. Precisely, we find that each fragment's estimated length is within 1.5% of the fragment's actual length (G.K.-S. Wong, pers. comm.) as determined from sequence data for the mapped region.
2. The method used to solve the resulting system of linear inequalities on the variables l_i, r_i, and x_j^k finds solutions that meet the equality portions of as many constraints as possible. We would prefer that $r - l = L$ whenever it is possible to draw the map with this equality. Hence, by choosing the bounding values as close to the observed experimental values as possible, we constrain the system as tightly as

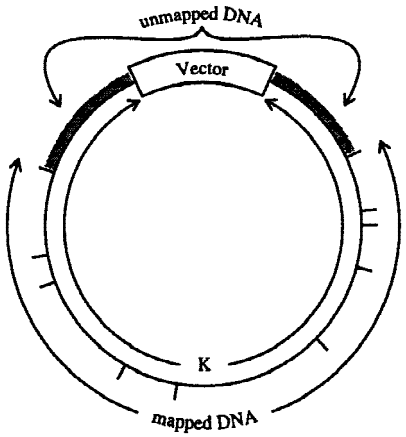

Figure 1 Schematic of a circluar clone showing the mapped and unmapped portions of the clone's insert for a particular restriction enzyme. The restriction sites correspond to the line segments perpendicular to the circle of the clone. The length *K* is the enzyme-specific clone length.

seems reasonable. In turn, we expect the solution returned by the algorithm to satisfy constraints closely, with deviations from the measured fragment lengths arising from the interactions of several constraints in the system.

The fudge factor α admits the possibility that the expected error put into the constraint system might be locally determined, in which case α would no longer be constant across the map and may vary based on user defined assessments of the ''quality'' or reliability of the map in different regions. More slack could be given to the linear constraints in poor quality regions, whereas those places where the quality was determined to be higher could be constrained so the fragment lengths stay closer to their experimental values. Obvious quality measures might include clone coverage or fragment size variance within the samplings of the same map fragment. We have not implemented this approach because of the great success experienced without the need to go to this level of specificity. Undoubtedly, this success rests on the accuracy of the fragment sizing and the strictness with which DNAM assembles the MCD maps. When poorer quality data is input though, this extra flexibility may allow a program like ATLAS more sensitivity to local weaknesses in the map, thereby permitting the program to make dramatic changes in poor quality map regions while preserving regions of higher quality.

Group Length

Though the goal in constructing any MCD map is to order the fragments in each domain, it occasionally happens that some fragments remain unordered with respect to each other but are ordered relative to the other fragments in the map. Such collections of unordered fragments are referred to as *groups of unordered fragments* or simply *groups* following Gillett et al. (1996). We extend the reference *group of fragments* to include instances in which fragments are linearly ordered, in which cases each group contains a single fragment. Given a group of unordered fragments, the sum, denoted by G, of the virtual lengths of the group's fragments estimates the distance between the restriction sites that flank the group. If x_i represents the distance from **O** to the left-most restriction site bounding the group and x_{i+1} the distance to the rightmost restriction site bounding the group, then G estimates the difference of these two variables. As was the case for L and the clone lengths, G only approximates the length of the group, so we allow some variability in this difference by introducing a fudge factor α for the group length constraints. Hence, for each group of unordered fragments in the MCD map, we have the group length constraints

$$G - \alpha G \leq x_{i+1} - x_i \leq G + \alpha G \qquad (2)$$

in which α is a small positive number, generally set to α = 0.001.

Clone End Localization

Each clone end is known to lie in exactly one group of unordered fragments from each enzyme domain. This localization constrains where the clone may start or stop in the map. There is no direct information in the map that indicates where *inside* the group the clone starts or stops. Other information in the map may help to determine the location of the clone end, in particular, the expected clone length (including estimates of the amount of unmapped DNA in the clone) and the sizes of the groups containing the other end of the clone (see Fig. 2). Because each clone end is localized in all domains, the constraints arising from this localization are the most informative of all those given and serve as the local ''glue'' between the enzyme domains. In particular, all other betweenness relations discussed are, in some sense, internal to the objects, either single restriction fragments or the clones themselves. The clone ends are the only physical objects whose approximate position is determined across all enzyme domains and, hence, constitutes the only connection between the data for different enzyme domains. We use some specific examples to explain the linear inequalities that arise from this constraint.

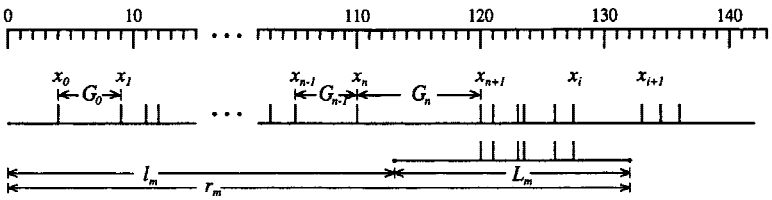

Figure 2 Our graphical representation for a single-enzyme restriction map consists of a horizontal line depicting the mapped target DNA with individual restriction sites indicated by vertical line segments as shown here. Once a choice for zero (**O**) is made, the distance from the left end of the map to any particular group boundary (or restriction site) can be determined and is represented by the variables x_i. The distances from zero to the left and right ends of each clone is represented by the variables l_m and r_m, respectively. In the formalism presented for the system of constraints on these variables, the length of each clone, denoted by L_m, as well as the length of each group of restriction fragments, G_k, are determined from the data available in each MCD map.

Suppose the left end of clone m is known to lie in a group that is bounded on the left end by the restriction site whose position is represented by x_n and is on the right by the restriction site represented by x_{n+1} (see Fig. 2). This constraint translates to the linear inequalities

$$x_n + \beta \leq l_m \leq x_{n+1} - \beta \qquad (3)$$

where β is again a small positive number but chosen in a slightly different manner

than the fudge factor used in the previous constraints. In practice, β = 100 bp is sufficient to push the clone ends away from the group boundaries except when a mapping enzyme cuts near the insertion points in the vector. Without the β-factor, the algorithm would place l_m at either of the two end points x_n or x_{n+1}. When clone ends are known to lie in the interior of the groups that contain them, β ensures that each clone end lies properly in the interior of the group to which it is localized. Similar inequalities hold for the position of the right end of the clone, r_m, in relation to the group boundaries flanking it.

Size of the Linear System

The number of variables $\{l_i, r_i, x_j^k\}$ and inequalities (from constraints 1, 2, and 3) can be expressed in terms of the number of clones, N, the total number of group boundaries, B, across all enzyme domains, and the number of enzyme domains, M. From our description above it follows that the number of variables is $2N + B$ and the number of inequalities is $2N + 2(B - M) + 2(2N) = 6N + 2(B - M)$ where the terms on the left-hand side are ordered according to the order of the descriptions of the constraints given above. Assuming a 1-Mb MCD map built from cosmid size clones (40 kb), three six-cutter enzymes, and 12× coverage, the number of variables is ~1330 and the number of inequalities is 3260. Increasing the coverage to 25× results in 1980 variables and 5200 inequalities. Systems of linear inequalities containing this order of variables and inequalities are easily handled by the algorithms described later in this paper.

Finding Solutions in the Presence of Inconsistent Inequalities

Anytime one is dealing with a large system of inequalities, the possibility that some contradiction will arise, between either two constraints directly or through the aggregation of the consequences of a subset of the constraints, must be borne in mind. In the case of the MCD maps, where we anticipate missing small DNA fragments and potential data or map assembly errors, the existence of contradictory inequalities is expected. Assuming no data and mapping errors, it is reasonable to expect that the effect of the missing small DNA fragments may accumulate in some regions and be expressed as incompatible constraints. Given that data and mapping errors also occur, the presence of these inconsistencies is a given and, in many instances, may indicate places in the map where more careful inspection is required to build the most accurate map possible given the data.

A Concrete Example of an MCD Map Inconsistency

Before making a general statement about the form of the inconsistencies encountered, we examine a concrete example taken from an MCD map of a yeast artificial chromosome (yWSS1980) from human chromosome 7. This map was constructed using cosmids as the subcloning vehicle and three restriction enzymes, *Eco*RI, *Hin*dIII, and *Nsi*I. When ATLAS initially examined the map, an *Eco*RI fragment measuring 568 bp was stretched to be 1000 bp, that is, an ~50% change. Four clone ends were found to lie in the 568-bp fragment, and each clone was re-examined for mapping or data errors. The gel images for clone g1980a113 revealed that a 568-bp fragment had been missed. Once this additional fragment was added to the fingerprint data for g1980a113 and the map was reassembled, ATLAS only stretched the 568-bp fragment by 0.1%, which is precisely the amount expected if no inconsistencies are present, because the Bellman–Ford algorithm prefers tight constraints as represented by the equality portions of the inequalities. Next, we explore how ATLAS detected this error, by way of explaining the inconsistent inequalities it detected and the underlying incompatible physical constraints implied by the initial MCD map.

Clone g1980a113 initially contained the *Eco*RI fragments measuring 2640, 4870, 2099, 2250, 914, 2140, 2880, and 4905 bp in this order from left to right. These fragments yield a minimum length for any clone containing them of 22,698 bp. The maximum allowable clone length as determined from the map includes the two complete fragments in which the left and right ends of the clone have been positioned. In Figure 3 these fragments correspond to the fragments bounded by x_1, x_2, and x_{10}, x_{11}. In the consensus map, the fragment containing the clone's left end measured 17,765 bp, and the fragment containing the right end measured 568 bp. The latter of these two fragments is the same fragment mentioned earlier as experiencing a

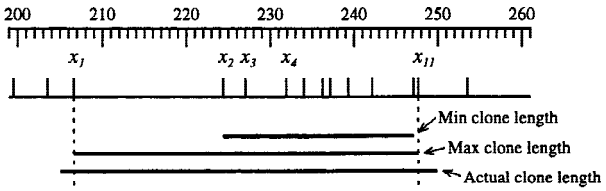

Figure 3 (*Top*) The *Eco*RI portion of the MCD map; (*bottom*) line segments showing both the minimum and maximum clone lengths allowable for this clone according to the *Eco*RI data. Below these two possible clone lengths, the actual clone length as an average of the clone's lengths from all three enzyme domains is shown. This is obviously too large a clone to fit between the flanking *Eco*RI restriction sites. Each variable is located directly above the restriction site with which it corresponds, omitting those that are too close together to be distinguished from each other. g1980a113's left clone end has been positioned between x_1 and x_2, and its right end lies between x_{10} and x_{11} that bound the 568-bp fragment. The inequalities representing this physical inconsistency are explained in the text in terms of these variables.

large relative stretch. These additional fragments indicate a maximum allowable clone length of 41,031 bp. Summing the lengths of all mapped fragments together with the unmapped DNA lengths contained in g1980a113 in each of the three enzyme domains produced domain-specific insert length estimates of 43,657 (*Eco*RI), 46,570 (*Hin*dIII), and 44,303 (*Nsi*I), so that the clone's length (average insert length) is 44,843.3 bp. Clearly this clone does not fit in the map at the position where it has been placed by DNAM, but this is only obvious when one considers the unmapped fragments in the clone, something which is not currently done in DNAM.

This inconsistency appeared in ATLAS as the following set of inequalities:

$$\begin{array}{lll} x_1 - l \leqslant -100 & x_2 - x_1 \leqslant 17{,}782.3 & x_3 - x_2 \leqslant 2{,}643.1 \\ x_4 - x_3 \leqslant 4{,}875.5 & x_5 - x_4 \leqslant 2{,}100.8 & x_6 - x_5 \leqslant 2{,}252.0 \\ x_7 - x_6 \leqslant 915.4 & x_8 - x_7 \leqslant 2142.6 & x_9 - x_8 \leqslant 2{,}882.9 \\ x_{10} - x_9 \leqslant 4{,}910.6 & x_{11} - x_{10} \leqslant 568.7 & r - x_{11} \leqslant -100 \\ l - r \leqslant -44{,}888.1 & & \end{array}$$

$x_1 - l \leqslant -100$ and $r - x_{11} \leqslant -100$ arise from the clone end localization constraints; $l - r \leqslant -44{,}888.1$ from the lower bound on the clone length constraint; and the remaining inequalities are the upper bounds on the group length constraints. The bounds are real valued rather than integral to reduce possibility of round-off errors propagating in large sums. Summing the first four rows of inequalities above gives the equation $r - l \leqslant 40{,}873.9$ that indicates the maximum clone length allowable in this position of the map as measured by the fragments covering the clone including the fudge factors described for each constraint. The remaining inequality $l - r \leqslant -44{,}888.1$ (or rewritten: $r - l \geqslant 44{,}888.1$) shows the minimum length expected for this clone as measured in the clone itself. Summing these last two equations, or equivalently, summing all thirteen inequalities directly produces the impossible inequality $0 \leqslant -4{,}014.2$ that is the form in which ATLAS found this inconsistency. This only indicates how ATLAS detected this inconsistency and not how it was resolved. This will be addressed after the discussion of the general algorithm used to solve the linear system.

The inconsistency described above is of a specific type because it involves a single clone and restriction sites from a single enzyme. Most inconsistencies are more complex. Despite its uniqueness, it demonstrates the general nature of all inconsistencies encountered. Specifically, although the inconsistencies arise from basic physical constraints that are violated, they do not become apparent until many different constraints are considered together. Most of the inconsistencies detected by ATLAS are not so simply interpreted and can involve many clone ends and restriction sites from all enzyme domains. Motivated by this specific example, we give a general description of the inconsistencies encountered by ATLAS.

General Structure of Inconsistencies

All inequalities in the system can be rewritten so they are in the form:

$$y_j - y_i \leqslant w(i, j) \tag{4}$$

in which y_i and y_j represent a pair of variables $\{l_m, r_m, x_h^k\}$ in the system and $w(i, j)$ is a numerical value representing either an upper or lower bound as defined in equations 1, 2, or 3. Written in this form, all inconsistent contraints appear as subsets of n inequalities,

$$\begin{array}{c} y_1 - y_2 \leqslant w(2, 1) \\ y_2 - y_3 \leqslant w(3, 2) \\ \vdots \\ y_{n-1} - y_n \leqslant w(n, n-1) \\ y_n - y_1 \leqslant w(1, n) \end{array} \tag{5}$$

such that $\Sigma\, w(i, j) < 0$. The inconsistency is apparent because the sum of all n inequalities (equation 5) yields $0 \leqslant \Sigma\, w(i, j)$. Consequently, no values can be found for $y_1, \ldots, y_n$ that will simultaneously satisfy all n of the constraints associated with these inequalities.

General Solution in the Absence of Inconsistencies

Any system of inequalities that can be rewritten as we have in equation 4 is referred to as a *system of difference constraints* and can be translated to a weighted directed graph **G** defined as follows:

1. The vertices of **G** correspond in a one-to-one manner with the variables in the system of difference constraints, $v_i \leftrightarrow y_i$, except potentially for an additional vertex s called the *source vertex* (which may need to be added if some vertices are unreachable from others).
2. **G** contains a directed edge (v_i, v_j) from v_i to v_j with weight $w(i,j)$ for each linear inequality $y_j - y_i \leqslant w(i,j)$. The direction of the edge is represented by an arrow pointing to v_j.

Figure 4 depicts the directed graph associated with the following system of difference constraints

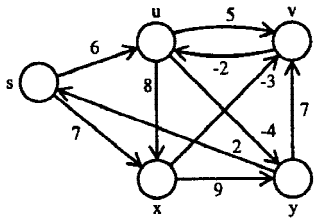

Figure 4 Associated to each system of difference constraints is a weighted directed graph called the constraint graph. The directed graph shown here corresponds to the system of difference constraints given in equation 6. The weight of each inequality is shown next to the corresponding directed edge, so that the inequality $u - s \leqslant 6$ corresponds to the directed edge joining s to u with weight 6.

(Cormen et al. 1990) involving the variables u, v, x, y, and s:

$$\begin{array}{lll} v-u\leqslant 5 & x-u\leqslant 8 & y-u\leqslant -4 \\ u-v\leqslant -2 & v-x\leqslant -3 & y-x\leqslant 9 \\ v-y\leqslant 7 & s-y\leqslant 2 & u-s\leqslant 6 \\ x-s\leqslant 7. & & \end{array} \quad (6)$$

The weight of a path $\langle s, v_{i_1}, v_{i_2}, \ldots, v_{i_{n-1}}, v_{i_n}\rangle$ in **G** is the sum of the weights along the edges covered by the path and hence corresponds to the sum of the inequalities associated with those edges. The sum of these inequalities yields a single inequality involving the variables associated with the first and last vertices on the path. For example in Figure 5a, the path $\langle s, u, v, u, y\rangle$ has weight 5 and corresponds to the inequalities

$$\begin{array}{ll} u-s\leqslant 6 & v-u\leqslant 5 \\ u-v\leqslant -2 & y-u\leqslant -4 \end{array}$$

which sum to $y - s \leqslant 5$. If s is assigned the value 0, then the weight of this path gives an upper bound of 5 for the variable y, which reflects the cumulative constraint placed on y by the inequalities along the path. If this method is used to assign upper bounds to all the variables along this particular path, then these values satisfy all the inequalities associated with the edges of the path. This is only a single path to y, and hence it is only one possible upper bound for y, but by considering all possible paths from s to y, we can find the smallest upper bound for the variable y. Because it is the smallest such upper bound, it satisfies all other constraints placed on y by inequalities in the system. For such a small graph, it is easy to find the shortest path to y from s. The path <s, x, v, u, y>, which with a weight of -2 yields the smallest upper bound for y. In general, the smallest upper bound for any variable y is exactly the length of the shortest path from the source vertex to the vertex that corresponds to y.

The difference system constructed from the MCD maps has an obvious source vertex s, any one of the leftmost clone ends in the map. Moreover, no additional edges from s are required because each vertex is reachable from this source as a result of the linear nature of the underlying model for the MCD map. In particular, each restriction site variable is related to the variables for the nearest restriction sites in that enzyme, and each clone end variable is joined to those restriction sites flanking the group in which the clone end is localized in all enzyme domains. So the variables in one enzyme domain are connected to the variables from other enzyme domains indirectly through the clone end variables.

For each variable y_j, the Bellman-Ford algorithm maintains an upper bound, denoted $\varepsilon[y_j]$, for the values of y_j (called a shortest path estimate) so that $\varepsilon[y_j] \geqslant a_j$ at all times in the algorithm in which a_j is the weight of a shortest path for the source s to v_j. Each upper bound for y_j will be determined by at least one inequality $y_j - y_i \leqslant w(i, j)$, and at each stage of the algorithm this inequality is kept track of in the form of a predecessor $\pi[j] = i$. The name ''predecessor'' originates from the directed graph in which v_i immediately precedes v_j on a path from s. Because of the manner in which $\varepsilon[y_j]$ and $\pi[j]$ are maintained throughout the algorithm, we have the following properties at any point in the algorithm:

1. $\varepsilon[y_j] - \varepsilon[y_{\pi[j]}] \leqslant w(y_{\pi[j]}, y_j)$ holds true whenever $\pi[j]$ is defined, and
2. the *precedessor subgraph* G_π that contains
   a. any vertex $v \in$ **G** with $\pi(v) \neq$ NIL together with the source vertex s
   b. any directed edge $[\pi(v), v]$ for which $v \in G_\pi$ and $v \neq s$

is a rooted tree with root s; that is, each vertex $v \in G_\pi$ lies on a directed path that starts at s.

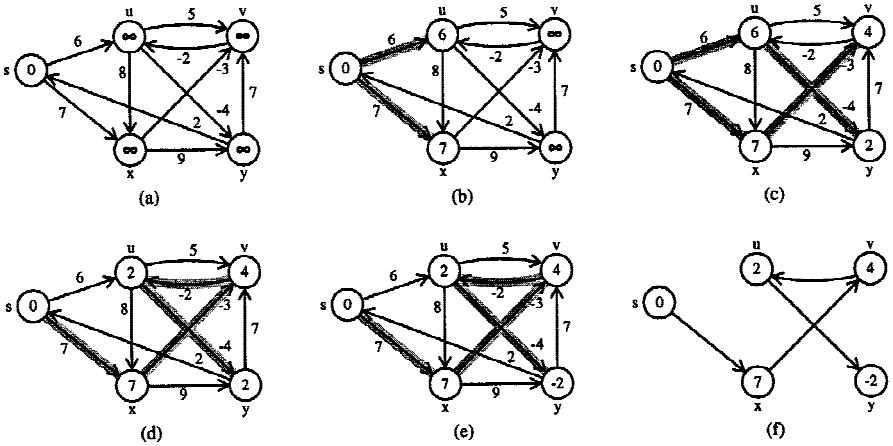

Figure 5 Representations of various intermediate stages of Bellman–Ford algorithm when applied to the simple system of difference constraints in equation 6. In this case, the source vertex is s, and the upper bounds ε for each variable, which measure the shortest path from s to the associated vertex, are displayed within the vertices. Predecessor relationships are indicated by the shaded edges, such that if $\pi(u) = s$, the edge from s to u is shaded. The relaxation process is applied to the inequalities (edges) in the order in which they appear in equation 6. (*a*) The graph after initialization and prior to the first round of relaxations; (*b–e*) the state of the estimates after each complete pass through all inequalities. The algorithm ultimately ends at the shortest path found in *e* where the shortest distance estimates are shown in each vertex. *f* simply highlights the final solution for this system; here, only the edges lying on the shortest path are displayed, and the distances to each vertex along that path are shown inside the vertices.

The algorithm begins by initializing $\varepsilon[s] = 0$, $\varepsilon[y_j] = \infty$, and $\pi[j]$ = NIL for all j, and choosing an arbitrary ordering of the vertices $(v_1, v_2, \ldots, v_{2N+B})$. Then, for each vertex v_j and each edge into v_j it adjusts the upper bound $\varepsilon[y_j]$ on the associated variable y_j so that it is equal to the smallest current path weight from s to v_j, given the current path weight estimates into the predecessors to v_j. The adjustment process is called the *relaxation step* and corresponds to taking an inequality $y_j - y_i \leq w(i, j)$ (edge into v_j) and the current estimates of $\varepsilon[y_j]$ and $\varepsilon[y_i]$ and if $\varepsilon[y_j] > \varepsilon[y_i] + w(i, j)$ the relaxation lowers $\varepsilon[y_j]$ to $\varepsilon[y_i] + w(i, j)$ and updates the predecessor $\pi[j]$ to equal i. Otherwise, relaxation leaves all variables unchanged. It is obvious that following the relaxation of $y_j - y_i \leq w(i, j)$, $\varepsilon[y_j] - \varepsilon[y_i] \leq w(i, j)$. Also, if $\varepsilon[y_j] = a_j$ before relaxation, then $\varepsilon[y_j] = a_j$ after relaxation and, hence, from that point on in the algorithm. This follows from the definition of a_j as the smallest upper bound, so to find $\varepsilon[y_j] < a_j$ implies a new path to v_j with a weight less than the smallest possible weight of all paths to v_j. Contradicting the definition of a_j. It can be shown inductively that after making $2N + B$ passes through the vertices, the algorithm converges to the solution $\varepsilon[y_j] = a_j$. Though this number of iterations is needed in the inductive proof of convergence, it is an upper bound representing the worst possible case. Many fewer iterations are actually needed in practice before the solution is discovered, and when a complete pass is made in which no edge is relaxed, the solution has been found and the algorithm terminates.

Figure 5 depicts the directed graph associated with the system of difference constraints given in equation 6 and shows the resulting directed graph, predecessor tree, and shortest path estimates after each pass through the vertices. On completion of the algorithm, we have the solution $(s, u, v, x, y) = (0, 2, 4, 7, -2)$.

Resolving Inconsistent Subsets of Inequalities

Because we seldom encounter MCD maps without some inconsistencies and the Bellman–Ford algorithm will not solve such systems, we developed a heuristic approach that modifies the system's inequalities in such a way that local inconsistencies are resolved one at a time until eventually a new system of difference constraints is obtained that is solvable. Ideally, this new system is very near the initial system constructed from the MCD map, and the degree to which it deviates is taken as a direct indication of the reliability of the map assembly process. We begin by exploring the concrete example arising from changing $x - u \leq 8$ to $x - u \leq 4$ and adding another variable t and the two inequalities $t - u \leq 4$ and $v - t \leq -7$ to the system in equation 6, which creates the inconsistent subsystems:

$$t - u \leq 4 \quad u - v \leq -2 \quad v - t \leq -7$$
$$x - u \leq 4 \quad v - x \leq -3 \quad u - v \leq -2$$

involving the vertices $\{u, v, t\}$ and $\{u, v, x\}$.

Using the same vertex ordering as in the system in Figure 5, with the additional vertex t placed last in the ordering, we can apply the Bellman–Ford algorithm as it was described above. During the third pass through the edges, when the edge (v,u) is relaxed, the predecessor to vertex u changes from s to v, which creates a loop, or cycle, in the predecessor tree, and violates the condition that the predecessor subgraph is a rooted tree with the source vertex s as the tree's root. Notice that the sum of the edge weights around this cycle is a negative value, in particular, -5.

Cycles whose edge weights sum to a negative number are referred to as *negative cycles*, and these clearly

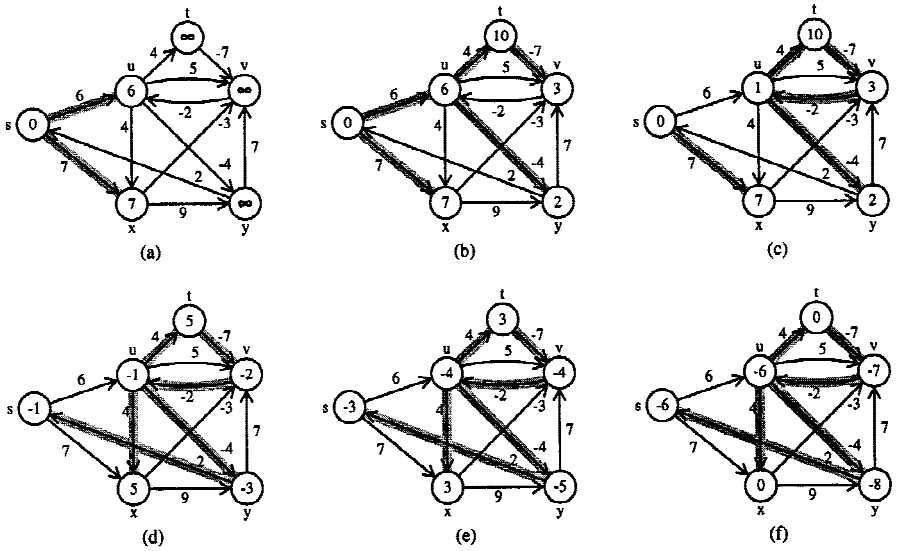

Figure 6 So as to demonstrate how the standard Bellman–Ford algorithm fails when inconsistent subsystems exist, one vertex and two edges are added to the previously solvable difference system. This difference system contains two negative cycles, which prevent the Bellman–Ford algorithm from solving the single-source shortest path problem. Our display method is identical to that used in Fig. 4, with the exception that the initial upper bounds (∞) are not shown. Instead, *a* depicts the state of the graph after the first pass of relaxation steps. Notice that after *c*, at which point *s* is no longer the root of the predecessor tree, the upper bounds for the variables *u*, *v*, and *t* continue to decrease. And after this point, any relaxation of one of the edges joining these vertices continues to lower the shortest path estimates without a lower bound in sight.

correspond directly to the inconsistent subsystems in the original system of difference constraints. Recall that the Bellman–Ford algorithm only converges in the absence of inconsistent subsystems, and by continuing to apply the relaxation steps to this example graph, we see why the algorithm fails to converge. The shortest path estimates for u, v, and t continue to decrease with each successive relaxation, and from the perspective of the shortest paths, this happens because each loop through these vertices decreases the "cost," or length, of any path containing these vertices, so that no *shortest* path exists. Given any candidate shortest path, one need only traverse the cycle $<u, v, t, u>$ one additional time to generate a shorter path. Hence, it is clear that there can be no shortest path from the source vertex to any vertex on a negative cycle. On the other hand, any vertices that are unreachable from a vertex on a negative cycle can still have a shortest path originating at the source vertex. Unfortunately, in the associated constraint graph for an MCD map, each vertex is reachable from any other vertex, so any negative cycle makes it impossible to estimate shortest paths. Hence, the standard Bellman–Ford algorithm will not solve our problem, and a modification to the algorithm is required.

Our first modification of the standard Bellman–Ford algorithm takes place in the relaxation step and allows us to determine if the predecessor subgraph has lost its rooted property as a result of the current relaxation. In particular, if an edge into any vertex v is relaxed and its predecessor, $\pi(v)$, changes during the relaxation, we trace the branch of the predecessor tree that contains v either to the source vertex (the tree's root) or until we encounter the original vertex v again. If we fail to reach the source vertex and find instead that we have followed the predecessor branch back to v again, ATLAS halts the Bellman–Ford algorithm and enters into the negative cycle resolution process, which we describe next.

If all errors encountered are owing solely to mis-sized fragments or missing small fragments, then there is a natural way to interpret the inconsistencies. Consider the example in Figure 6 in which the edges $t - u \leq 4$, $v - t \leq -7$, and $u - v \leq -2$ occur in a negative cycle of weight -5, and imagine that the edge weights arose from mis-measured fragment sizes. Because mis-sizing is generally made in the determination of the relationship between fragment size and mobility in the gel, the errors tend to be proportional to the measured fragment size. So, if we had to guess a priori at which edge weight along the negative cycle deserved to be modified by the largest amount, the (t,v) edge with a weight of -7 would stand out. Recall that the negative edge weights arise because the inequality $v - t \leq -7$ was originally a lower bound, that is, $t - v \geq 7$, so the weight still reflects some physical measurement in the MCD map. Because the negative cycle's weight is -5, adding 5 to the edge weight -7 would eliminate the negative cycle, because the cycle's weight would then be 0. In-

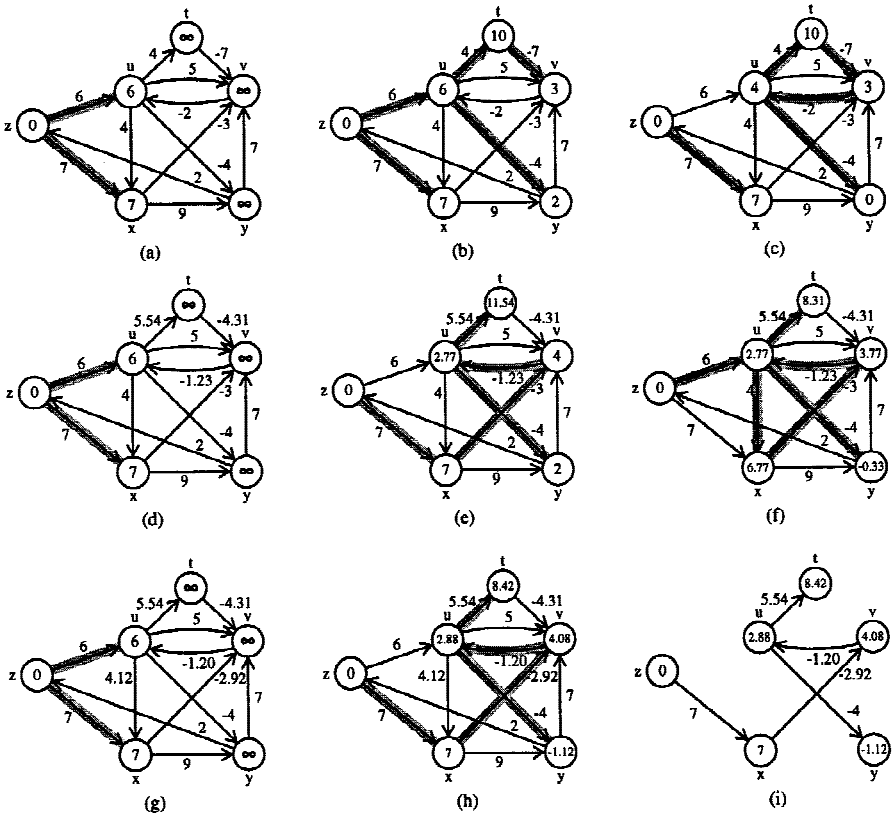

Figure 7 The individual steps of the modified Bellman–Ford algorithm are shown using the same graphical devices used in Figs. 5 and 6. *a–c* correspond to the first three relaxation passes of the Bellman–Ford algorithm. Once ATLAS detects the presence of the negative cycle $<u, v, t>$ Bellman–Ford is halted, and the edge weights along the negative cycle are increased so as to eliminate the negative cycle. (*d*) The new edge weights. *d–f* represent the first three relaxation passes of Bellman–Ford on the modified constraint graph. Again, ATLAS detects the negative cycle $<u, v, x>$ after three relaxation steps and Bellman–Ford is halted and the edge weights along the cycle are increased to eliminate the negative cycle. (*g*) The new constraint graph resulting from this modification. (*h*) The final solution given by Bellman–Ford for the constraint graph represented in *g*.

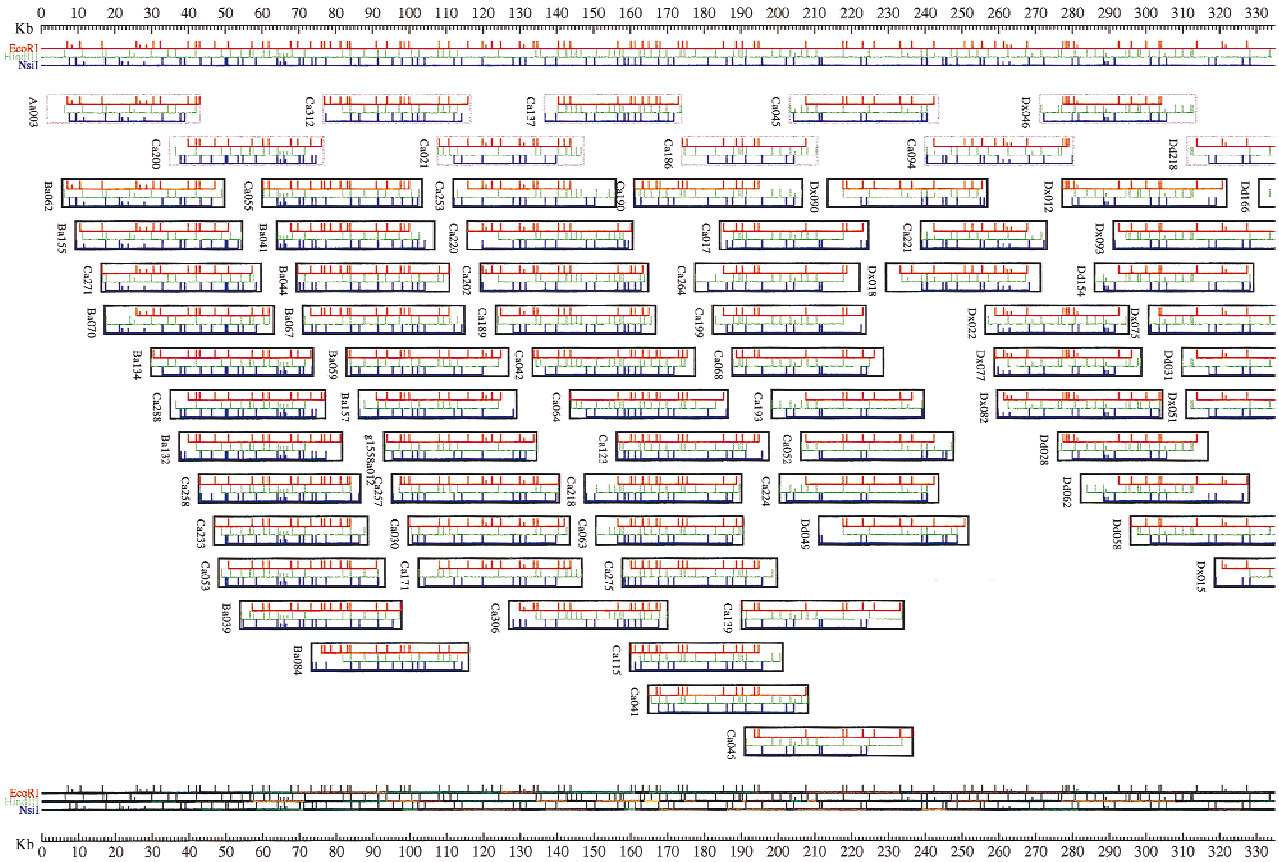

Figure 8 Depiction of one of several ways an individual user may view the end result of the ATLAS program and attempts to combine both the linear diagram for the MCD map being considered and the new difference system developed by ATLAS that is free of negative cycles. The restriction fragments are drawn for each enzyme map as horizontal lines bounded by restriction sites (vertical line segments). Group boundaries are indicated by taller vertical site marks, with the half-height site marks representing unordered fragments in the group. Inside the groups, the fragments are arbitrarily ordered by increasing fragment size. Clones are represented by rectangular boxes inside which the clone's restriction fragments are drawn in alignment with the map. The clone boxes often extend past the ends of the first and last fragments in the clone reflecting total clone insert length that is factored into ATLAS's clone end localization calculations. Because ATLAS modifies the bounds in the system of inequalities (which corresponds to changing both fragment and clone lengths), the final picture may contain fragments that have been stretched or shrunk in length from those in the actual MCD map. These changes often occur in regions of the map that contain mapping errors or may occur in regions in which many small target fragments have been missed. The duplicate MCD map drawn below the clones is color coded according to the percentage change in fragment length needed for ATLAS to draw the picture. Bright orange and green indicate that the fragments were either stretched or shrunk by 15% of their original length.

stead of modifying a single edge along the cycle, ATLAS distributes the absolute value of the negative cycle's weight, here 5, proportionally across all the edges of the cycle according to the formula

$$\tilde{w}(i,j) = w(i,j) + |w(i,j)|\frac{|w(\gamma)|}{\Sigma_{\text{edges of }\gamma}|w(k,l)|} \qquad (7)$$

in which γ is the negative cycle and $w(\gamma)$ its weight. So that in our example, the new inequalities would be

$$t - u \leq 4 + 4 \cdot \frac{5}{4+7+2} = 5.54$$

$$v - t \leq -7 + 7 \cdot \frac{5}{4+7+2} = -4.31$$

$$u - v \leq -2 + 2 \cdot \frac{5}{4+7+2} = -1.23$$

Once the detected negative cycle has been resolved, ATLAS begins the Bellman–Ford algorithm again. Notice that for the example of Figure 6, the second negative cycle has not been encountered yet, but the resolution of the first negative cycle has lessened the amount the edges must be modified in the second negative cycle because the weight of edge (v,u) has been increased and this edge lies on both negative cycles (see Fig. 7). ATLAS will detect this negative cycle on the next run of the Bellman–Ford algorithm and correct it. Figure 7 demonstrates the evolving difference constraint that ATLAS constructs and then solves. Because the edge weights are always increased, no new negative cycles will be created solely by resolving other negative cycles. This observation ensures that the modified algorithm will terminate in finite time, but

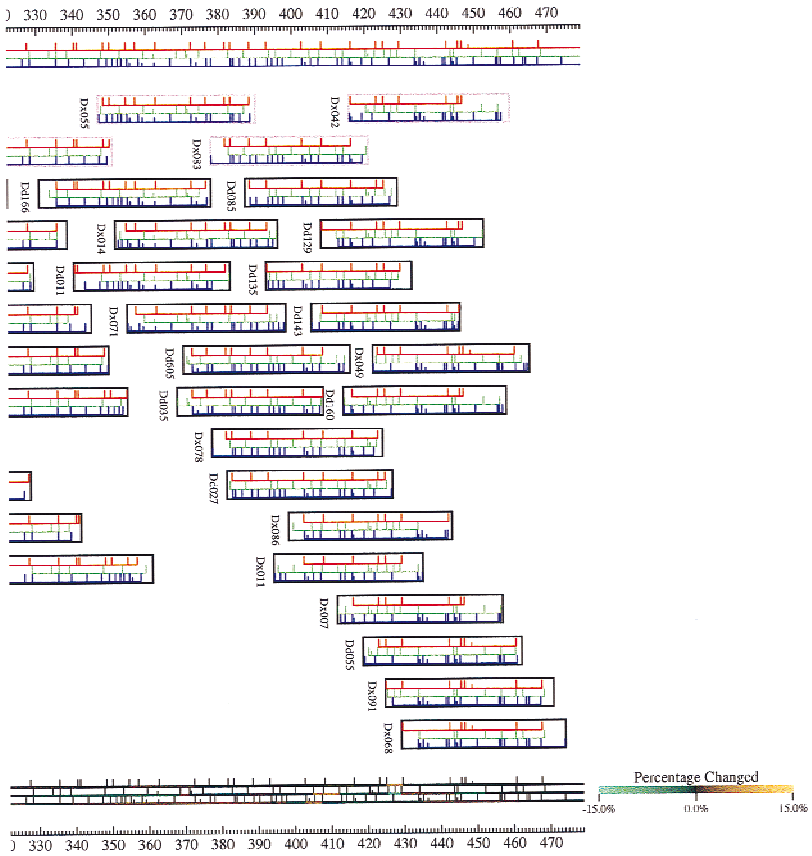

Figure 8 (*Continued*)

the length of time clearly depends on the number of negative cycles present in the MCD map. Notice that ATLAS is modifying the constraint system to produce a solvable one, and in the final pass through Bellman–Ford it finds the solution for this new system. So in the end, ATLAS creates two pieces of data: a solvable system of difference constraints and the solution to this new system.

Whereas the original Bellman–Ford algorithm arrives at the same answer independent of the vertex ordering, the modified algorithm's solution depends on the vertex ordering because this determines the order in which negative cycles are detected and resolved. Though there are many possible vertex orderings one may consider, two orderings occur naturally owing to the linear nature of the maps. ATLAS starts with the obvious left-to-right vertex ordering derived from the partial ordering of the associated map objects, placing those clone ends that appear in the same fragment together in the list one after the other arbitrarily. Hence, the negative cycles located at the left end of the map are detected first. If one reversed the ordering, ATLAS would detect inconsistencies near the right end of the map before those near the left end. To reduce potential differences between the left-to-right and the right-to-left orderings of the vertices, ATLAS reverses the vertex ordering after each negative cycle is detected and resolved.

Figure 8 shows a sample of ATLAS's output for an MCD map from human chromosome 7. The map is drawn across the top of the display, and each clone (subclones are not shown) is represented by a rectangle inside which the clone's fragments are drawn in alignment with the map's fragments. The first two rows of clones (outlined with purple) were selected to be sequenced based on a minimal mutual overlap criterion using the computed values from ATLAS. Note that some of the clones extend past the ends of the fragments they contain, reflecting ATLAS's estimation of the amount of unmapped DNA in those clones. Because the system of inequalities solved by ATLAS may not be exactly the same as that initially derived from the MCD map, ATLAS needs to communicate where and by how much it had to change each edge length. This information is conveyed by the additional copy of the map across the bottom of the picture. Here, rather than displaying each enzyme map in its own color, each fragment in each domain is colored to reflect the percentage by which it was changed to solve the system of inequalities. Many more examples of ATLAS-generated MCD map images can be found on the World Wide Web at URL http://www.genome.washington.edu/UWGC/.

METHODS

ALTAS is implemented in the form of two programs: Atlas_i and MapBrowser. Atlas_i implements the algorithmic process described above. Specifically, it reads the DNAM map file, constructs the system of linear constraints described above, and applies the modified Bellman–Ford algorithm. The DNAM map file contains a partially ordered set of clone ends and restriction fragments for each enzyme domain mapped. On completion, Atlas_i outputs a list of new fragment lengths for each domain and the positions of each clone end as determined by the solution of the modified linear system. Often, these Atlas_i-computed positions allow the partial ordering of these objects to be refined to a linear ordering.

MapBrowser is a perl/Tk graphical user interface that parses the new MCD map coordinate data output from Atlas_i and presents it to the user in the form of a graphical representation. Once Atlas_i completes its computation of the map object's coordinates, these are fixed, but there are many ways the user may want to interact with the MCD data. The most critical feature of this display is the assessment of the percent error in each group's length. A group's percentage error is the

percent difference between the group's virtual length (as determined by DNAM) and that group's Atlas_i length (as measured from the coordinates of the restriction sites bounding the group). MapBrowser displays the percentage error for all groups in the map using a pseudo color display of the map. This allows the mapper to quickly isolate regions in the map that are potentially in error but were undetectable from the perspective taken by DNAM. MapBrowser also allows access to any data contained in the MCD map via a simple mouse driven, graphical interface.

ACKNOWLEDGMENTS

The publication costs of this article were defrayed in part by payment of page charges. This article must therefore be hereby marked "advertisement" in accordance with 18 USC section 1734 solely to indicate this fact.

REFERENCES

Cormen, T.H., C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to algorithms*. The MIT Press and McGraw-Hill Book Company, Cambridge, MA.

Fasulo, D., T. Jiang, R.M. Karp, R. Settergren, and E.C. Thayer. 1996. An algorithmic approach to multiple complete digest mapping. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, pp. 118–127. ACM Press, New York, NY.

Green, E.D. and P. Green. 1991. Sequence-tagged site (STS) content mapping of human chromosomes: Theoretical considerations and early experiences. *PCR Methods Appl.* **1:** 77.

Gillett, W., L. Hanks, G.K.-S. Wong, J. Yu, R. Lim, and M.V. Olson. 1996. Assembly of high-resolution restriction maps based on multiple complete digests of a redundant set of overlapping clones. *Genomics* **33:** 389–408.

Jiang, T. and R.M. Karp. 1998. Mapping clones with a given ordering or interleaving. *Algorithmica* **21:** 262–284.

Olson, M.V. and E.D. Green. 1990. Chromosomal region of the cystic fibrosis gene in yeast artificial chromosomes: A model for human genome mapping. *Science* **250:** 94–98.

Olson, M.V. and P. Green. 1993. Criterion for the completeness of large-scale physical maps of DNA. *Cold Spring Harbor Symp. Quant. Biol.* **58:** 349–355.

Wong, G.K.-S., J. Yu, E.C. Thayer, and M.V. Olson. 1997. Multiple-complete-digest (MCD) restriction-fragment mapping: Generating sequence-ready maps for large-scale DNA sequencing. *Proc. Natl. Acad. Sci.* **94:** 5225–5230.

Received September 10, 1998; accepted in revised form December 1, 1998.