CHUNK GRAPH WALKER DESIGN (draft)

Overview

The goal of chunk graph walking is to find a consistent set of paths through the chunk graph which maximizes the inclusion of chunks and essential overlap edges while minimizing the number of clone mate constraints which are violated. The chief strategy will be to use clone mates to navigate the chunk graph. This means that clone mate information will be used to choose between alternate branches in the chunk graph for any given path. The chunk graph will be extended to include mate edges between every pair of chunks which contain one member each of a clone mate pair. There will not exist a mate edge for every clone mate pair because a consistent set of clone mate pairs between the same pair of chunks will be combined into a single weighted mate edge. The mate edge encodes the number of clone mate pairs consistent with this edge, the implied orientation of the chunks (can use the same notation as for overlap edges), and the possible distance range between the chunks (this can be either an implied overlap or gap). The possible distance range is determined by taking the intersection of possible distance ranges for each clone mate pair contributing to the mate edge (the distance range for each clone mate pair is a result of the clone length distribution for a given library as initially defined to the system in a distance message). The reader should note that there may be more than one mate edge between a pair of chunks if the set of clone mate pairs between the chunks are not all mutually consistent. A key component of the strategy is that confirmed edges are highly reliable. A confirmed mate edge is a mate edge with at least two underlying clone mate pairs. An unconfirmed mate edge can also be confirmed by a consistent overlap edge in the underlying chunk graph. A further extension of confirmation is that finding a path of any combination of overlap and mate edges which is consistent with another unconfirmed mate edge confirms that mate edge.

At any point in the chunk graph walk the problem can be viewed as making a path extension decision in one direction based on the mate and overlap edges going in that direction from the current partial path. The algorithm for choosing the path extension will be primarily greedy relying on the assumption that confirmed mate edges are highly reliable. Unfortunately, the path extension algorithm also relies upon being able to label chunks as repetitive (used in more than one path) or unique (used in only one path) and this labeling is not as robust as confirmed mate edges. The initial labeling of the chunks will rely on a combination of arrival rate statistics within the chunk and type and position of branch points within the chunk. The arrival rate statistic gives a precise maximum likelihood ratio for the probability of a chunk being unique versus repetitive but the separability of this statistic is poor for short chunks and low coverage levels. The branch point patterns are more qualitative but help when the arrival rate is ambiguous. It is not clear what level of coverage will be necessary for this algorithm to have much success. It is also not clear how good the chunk labeling needs to be or what the ratio of false positive to false negative unique chunk labeling will be optimal for the algorithm.

The path extension algorithm uses the confirmed mate edges and chunk labeling in the following way: first only the mate edges out of unique chunks are used because repeat chunks can be on multiple paths and hence have mate edges to multiple paths which does not help us extend the current path with confidence, second only confirmed mate edges are used to extend the path because these are highly reliable, and finally, if confirmed mate edge conflicts arise then the algorithm attempts to backtrack and relabel the chunk causing the conflict from unique to repeat. The seeding chunk for path extension will be the chunk with the highest uniqueness score because the algorithm works best starting from correctly labeled unique chunks.

Each chunk will also be checked for internal clone mate consistency and suspicious chunks will be relegated to the end of the queue for path extension.

Memory Usage

The major memory consuming data structures will be for the extended chunk graph which includes chunks, overlap edges, and mate edges.

Interface

The input to the chunk graph walker consists of messages/data: clone mate messages specifying clone mate pairs, distance messages specifying the statistics of clone lengths for a given library, and chunk graph messages of types to be determined specifying the chunk graph (chunks and chunk overlap edges). The chunk information must specify the location within a chunk of every fragment (this is complicated if fragments are allowed to be in multiple chunks). In addition, information about chunk uniqueness such as the maximum likelihood arrival rate statistic and branch points must either be explicitly passed to or computed by the chunk graph walker.

The structure of the chunk messages is still being defined but more importantly the content of the chunk needs to be specified. The chunk could simply be defined by the set of essential edges from the overlap graph internal to the chunk. The chunk graph walker needs to have coordinate ranges for the endpoints of all the fragments making up a chunk (coordinate ranges due to the uncertainty of the final alignment). These coordinate ranges can be passed to or computed by the chunk graph walker.

The output of the chunk graph walker is a set of paths through the chunk graph. The chunk graph walker must also pass through any relevant messages to the next module such as clone mate messages. Currently, the next module is the repeat separator/resolver.

Design

For now it is assumed that the chunk graph builder is responsible for the following steps: maximum likelihood ratio of arrival rate statistic for chunk uniqueness labeling, branch point labeling of chunks, and coordinate range representation of fragment positions within chunks. This leaves three major tasks for the chunk graph walker: building the extended chunk graph by adding mate edges, checking for clone mate consistency within chunks, and finding paths through the chunk graph.

Building the extended chunk graph should be straightforward. For each pair of chunks which contain one member each of a clone mate pair construct a mate edge between these chunks. The mate edge specifies the orientation of the chunks with respect to each other and the distance range between the chunks. There may exist more than one clone mate pair between a given pair of chunks. In this case the clone mate pair must be checked for consistency with any existing mate edge. If it is consistent then the mate edge is modified to include the new clone mate pair, otherwise a new mate edge is created. Clone mate pairs are consistent for chunk pairs if they give the same chunk orientation and their distance ranges overlap. A special note is needed for fragments which may be placed in multiple chunks. These fragments only occur because they are contained in at least one other fragment and are part of a repeat. These fragments should not be used in the determination of mate edges because they violate the assumptions made later about mate edges (one such assumption is that all mate edges emanating from a unique chunk will terminate on chunks along a single consistent path with the exception of chimeric mate edges – we can obviously not rely on this being true for any mate edge based on a fragment which can be placed in multiple chunks).

Checking for clone mate consistency within a chunk is also straightforward. Each fragment within a chunk whose clone mate should lie within the chunk but does not is an inconsistency (this can be extended to include clone mates which should overlap with the chunk but do not). A threshold based on a probabilistic model of how often clone mate constraints should be violated due to chimeric clones will be used to label inconsistent clones. In addition, a different threshold can be used for two special cases: inconsistent clones in the same region and confirmed inconsistent clones (these are confirmed mate edges which should have an overlap edge between the same pair of chunks but do not).

The first step in constructing a path through the chunk graph can also be thought of as a verification of a chunk’s labeling as a unique chunk. This could in fact be an initial pass through all of the chunks if an efficient way to store and access the results is used so that the computation does not have to be repeated. This chunk verification is a check of the consistency of the mate edges emanating from a single chunk based on the assumption that the chunk is unique which infers that all nonchimeric mate edges should fall along a single path. In practice this means that all confirmed mate edges (highly unlikely to be chimeric) must form a consistent path and that there also not be too many unconfirmed mate edges (using a statistical model of chimeric clone mate pairs) that do not fall along this consistent path.

A path can be thought of in at least two ways: as an ordering (possibly partial) of chunks and/or a range of coordinates for each chunk relative to some initial chunk which implies an ordering. The initial path construction step/chunk uniqueness verification can be thought of as constructing a coordinate range representation of a path relative to the initial chunk based on mate edges incident to that chunk. We fix the starting coordinate of the initial chunk s_i equal to 0 and the final coordinate f_i to a range based on the length and length variation of the initial chunk. For each chunk j with a mate edge to initial chunk i we can calculate coordinates s_j and f_j based on the length and variation of mate edge m_{ij} and chunks c_j and c_i. We can sort this list of start and final coordinates for each chunk by the start coordinate and then quickly search if any chunks have a potential overlap by checking for each c_j with s_j and f_j if there exists a c_k with s_k where $s_j <= s_k < f_j -$ min_overlap. We can then quickly check for the existence of these potential overlap edges in the chunk graph. We will also need to explicitly check for overlaps missed by the overlapper module at this point. For each overlap found we will form an extended chunk (or echunk) which will have similar properties to a chunk but which will also have internal coordinates that specify where each constituent chunk falls within the echunk. The relative coordinates of the echunk within the path will be inferred from the constituent chunks relative coordinates.

Any chunks which had unconfirmed mate edges with the initial chunk c_i but were found to have at least one consistent overlap with another chunk are now considered to be confirmed as well (so by definition all echunks are confirmed). There is one important exception to this rule. If an overlap with the initial chunk c_i would not occur if the fragment reads which formed the unconfirmed mate edge were removed then this mate edge is still unconfirmed (and the echunk formed by this overlap will be dissolved). The reason for this is that there must be independent confirmation to label a mate edge as confirmed or in other words a single clone is not allowed to confirm a mate edge. We will now check to see if all confirmed chunks (chunks with confirmed mate edges to c_i) and echunks (confirmed via overlap) can be placed consistently along a single path (note we will not worry about conflicts with unconfirmed chunks unless the number of conflicts becomes too high). For the rest of this discussion we will not distinguish between chunks and echunks except when necessary and the words chunk and chunks should be considered to mean both chunks and echunks unless explicitly specified otherwise. The problem now is to see if a set of chunks with a given set of starting and final range coordinates and minimum length coordinates which are known to not overlap by more than some minimum overlap length can be placed (coordinates assigned) with out violating any of these constraints. This problem is easy if the coordinate ranges of all the chunks are never contained in another coordinate range – for c_i c_j if $s_i < s_j$ then $f_i < f_j$ because we can just start with the smallest s_j and for each s_k in increasing order shift the chunk c_k as far left as possible without causing an overlap we know does not exist until we have a consistent set of coordinates for all of the chunk or we violate the f_k constraint for some chunk. For contained coordinates where chunk c_k's coordinate range contains chunk c_j's coordinate range the choice is not so simple because there are four possibilities in trying to order the two chunks: first there may be no way to place the two chunks without violating the no overlap rule, second we may be able to place chunk c_k either to the left or the right of c_j, third we can place c_k only to the left of c_j, or fourth we can place c_k only to the right of c_j. We can proceed for the four cases as follows: case one we know we have a conflict and can quit, case two we choose the placement of c_k which results in the leftmost placement of the two chunks combined which should always be placing c_k to the left of c_j, case three and case four we have no choice since only one placement is valid. The real problem arises when we have nested contains and we need to keep track of all possible choices for case two in case we need to backtrack based on a later constraint. The way to implement this backtracking is to satisfy the most deeply nested constraints first and work our way up. In theory this could lead to an exponential explosion but in practice the amount of nesting should be minimal and case two nesting should be even rarer.

A better way to formulate the above algorithm is as a partial ordering on the chunks followed by a topological sort. First for each chunk c_i we see if c_i can possibly be placed before each chunk c_j based on their range coordinates $((s_i,f_i)$ and $(s_j,f_j))$, minimum sizes (m_i and m_j), and minimum allowed overlap (min_overlap). If $(s_i + m_i - \text{min\_overlap}) < (f_j - m_j)$ then we will define $c_i < c_j$. If when we compute this we find that neither $c_i < c_j$ nor $c_j < c_i$ then we have a conflict and cannot find a single consistent path. If we find both $c_i < c_j$ and $c_j < c_i$ then we cannot determine the relative order between these two chunks and we ignore both relations. Otherwise, either $c_i < c_j$ or $c_j < c_i$ but not both are true and we have defined a partial order on the chunks. We now proceed basically as outlined above, where we start with the leftmost chunk and shift it as far as possible to the left then shift the next chunk as far as possible to the left without producing an overlap conflict and make sure as we proceed that we do not have a coordinate range conflict. If no conflicts arise there does exist a single consistent path for this set of chunks. The only complication is when the topological sort does not produce a total ordering but has ambiguity between some chunks. For these unordered sets of chunks we need to find the leftmost unconflicted ordering. We could just check each possible permutation of the set of chunks and keep the leftmost unconflicted one. Preferably, we can predict an ordering of the permutations by which will give the leftmost result and then check them in this order for conflicts so that when we find an unconflicted one we can quit. Several rules hold and can be proved by contradiction. First some terminology, relabel the topologically unordered set of chunks from 1 to n such that $s_1 <= s_2 <= \ldots <= s_n$ and then use the same conventions as before and define $P = p_1p_2\ldots p_n$ to be a permutation of the chunks. We will say P is unconflicted if by doing the leftmost shifting test on P we do not have any overlap or range conflicts and we will call the resulting rightmost coordinate of this leftmost compaction P_f. For any unconflicted P where p_n not equal to c_n and $P_f <= f_n$ we can construct an unconflicted P' with $P'_f = P_f$. This is easy to see by just taking $p_k = c_n$ and constructing $P' = p_1p_2\ldots p_{k-1}p_{k+1}\ldots p_np_k$. We can slide $p_{k+1}\ldots p_n$ by m_n to left because no starting coordinate violations will occur because s_n is the largest starting coordinate for this set of chunks and $p_k = c_n$ so $p_{k+1}\ldots p_n$ will all have start positions $> s_n$. Now we just tack p_k onto the end which will not violate the final coordinate constraint for c_n because $P_f <= f_n$. Recognizing that any P ending in $p_n = c_n$ has $P_f <= f_n$, we can conclude that any P ending in $p_n = c_n$ is more leftmost than one that does not. By induction we can make the argument that you always want to try the next untried permutation which – well maybe not I'm still working on it. It is clear that if you can order $c_1c_2...c_n$ such that $s_1 <= s_2 <= … <= s_n$ and $f_1 <= f_2 <= … <= f_n$ then the identity permutation $P = c_1c_2...c_n$ always has the smallest P_f – the proof by contradiction is very similar to the one above.

There is one obvious decomposition of the problem which is to split the path anywhere where there is guaranteed to be a gap. Formally, if we order the chunks such that $s_1 <= s_2 <= … <= s_n$ then we can partition the problem into two subproblems $c_1c_2...c_j$ and $c_{j+1}c_{j+2}...c_n$ when for all f_i $i <= j$, f_i – min_overlap $< s_{j+1}$. Once you have decomposed the problem in this way as far as possible and for the subproblem again renumbered and ordered the chunks such that $s_1 <= s_2 <= … <= s_n$ there are some obvious constraints/bounds on being able to find an unconflicted solution. One bound is that s_1 + the sum of m_i over i = 1 to n – (n-1) * min_overlap < max (f_i). We can actually order the final coordinates in decreasing order and apply this bound recursively after removing the chunk with the largest final coordinate each time. This bound actually has to hold for any subset of the chunks and might be worth computing for every contiguous subset. The DAG implied by the partial order defined above without transitive edges removed can be used to compute this bound at all interior nodes for ordered subsets. We can try to decide on the tightest bound for any subset of chunks. One observation is that if s_1 + the sum of m_i over i = 1 to j – j * min_overlap < s_{j+1} then the bound s_{j+1} + the sum of m_i over i = (j+1) to n – (n-j-1) * min_overlap < max (f_i) is a tighter bound. In fact these bounds are just a way of formalizing what the leftmost test is. We can use these bounds and a similar one – if $c_i < c_j$ then $s_i + m_i + m_j$ – min_overlap $<= f_j$ – and the DAG to refine all of s_i and f_i for each chunk c_i. Conflicts would show up when $s_i + m_i > f_i$. Once the range coordinates are refined we may need to add new edges to the DAG to reflect pairs that used to be unordered but now can be ordered or we may even find pairs that used to be unordered but now conflict. We can then repeat doing the refinement and adding edges until we find a conflict or stabilize (note that we are guaranteed to stabilize in finite time since the ranges always are reduced and the number of edges which can be added to the DAG is finite). I cannot yet prove that if we stabilize without finding a conflict that there does exist an unbranched consistent path but I am starting to think this might be provable. If nothing else this algorithm seems worth pursuing just to be able to refine the range coordinates.

After some preliminary research the determination of a consistent path under the above formulation appears to be NP-hard/complete but I am not yet positive about the existence problem. What is also clear is that if a total or nearly total ordering can be determined then the problem is merely quadratic. I think a lot of our actual problems will be highly ordered and hence solvable but one way to make this even more true is to increase the number of chunks which are partially ordered with respect to each other. The first thing we can do is check for the existence of mate edges between pairs of chunks not involving the initial chunk which are consistent with the range coordinates. If one of these mate edges exist it will immediately give a partial ordering of the pair of chunks and might refine the range coordinates for the chunks as well. This approach is not without pitfalls because scenarios exist where the inference drawn may be incorrect due to one or both chunks being nonunique and the mate edge being consistent but not representative of the true path – our current thinking is that these scenarios have low probability and the gain in ordering out weighs the risk. The next step could be to find all chunks in the chunk graph which connect any pair of our current set of chunks consistent with the range coordinates. These very short paths in the chunk graph could be found very efficiently but we must remember they suffer the same pitfalls (not clear whether the risk is higher or lower than mate edges between pairs of chunks). Either of these can be extended to include longer path searches in the extended chunk graph to try to determine a partial order between pairs of chunks but the efficiency of the search will decrease and the risk might increase.

At this point I would plan to proceed with the two additional ordering steps given above and then construct the DAG for the partial order and iterate twice to refine the DAG as discussed above. If we have a total order with no conflicts we're done. If we have a conflict we're done. Otherwise we can use a simple heuristic to try to find an order that satisfies the DAG and the constraints and if we do we're done. Otherwise I would either assume there are no conflicts or try to search the extended chunk graph to get a better ordering. We can also choose to allow the extra ordering steps to confirm unconfirmed chunks.

We can extend each chunk as described above into what we will call a chunk's confirmed scaffold (note this is only for unique chunks) composed of a partial ordering of neighboring chunks and implied fuzzy coordinates with respect to the initiating chunk. At this point we could pursue a greedy approach by either extending scaffolds from the most unique remaining unextended chunk in the scaffold or by merging scaffolds based on a similar criterion. A more elegant and perhaps insightful approach is to treat the scaffolds as elements which can be overlapped based on their unique chunk content. Once we do the overlapping we can produce an overlap graph and collapse this into a chunk graph for the scaffolds. If our uniqueness labeling is correct and discounting structural polymorphisms and chimeric confirmed mate edges there should be no branches in this chunk graph. The presence of branches would alert us to regions where we need to look for problems in our assumptions or the data.

The first challenge is to define what an overlap is between scaffolds. A minimum requirement is that the two scaffolds have at least one unique chunk in common and this condition should make the search for potential overlaps efficient. The second requirement is that the chunks the two scaffolds share have the same relative orientation and ordering in both scaffolds. These two requirements may be sufficient but at least two others should be considered: that the relative distances between shared chunks be approximately the same and that merging the two scaffolds does not result in any obvious conflicts as defined previously. In any case, the two scaffolds should be merged and links between unshared chunks in the two scaffolds should be used to help refine any ambiguities in the partial orders of the two scaffolds and confirm any unconfirmed mate edges. There are two possibilities for characterizing the implied overlap of the merged scaffolds: the length of the overlap and overhangs can be estimated from the fuzzy coordinates which will also imply the type of the overlap (contained, normal/prefix/suffix dovetail), or just the implied partial order of the merged scaffolds can be used to determine just the type of the overlap. Due to the nature of scaffolds and unique chunks being used to determine overlaps we do not expect that there can be multiple legitimate overlaps between the same pair of scaffolds and will not allow these to occur. Under this assumption the type of an overlap alone is sufficient to perform transitive edge removal. One concern however is that the gaps inherent in the scaffolds if large enough to contain another scaffold or part of a scaffold can lead to branches in the collapsed graph which only exist because real overlaps between scaffolds are not detected due to these gaps.

Statistics and Testing

This section delineates statistics that can be collected and tests that can be made especially for simulated data that can be used to assess the potential success of walker strategies and the correctness of clone mate information and walker algorithms. The statistics will mostly involve clone mates/mate edges since that is the primary basis of the walker algorithms. There are some obvious statistics to provide histograms for on a per chunk edge basis and possibly a per chunk basis which can be computed for the enhanced chunk graph before walking commences: internally consistent chunks, confirmed mate edges, unconfirmed mate edges, all mate edges, sum of chunk sizes for confirmed/unconfirmed/all mate edges, maximum extension off of the edge of a chunk via confirmed/unconfirmed/all mate edges, and ratio of extension filled (sum of chunk sizes divided by maximum extension) for confirmed/all chunk edges. These statistics would be grouped into several classes such as true unique chunks, true repeat chunks, chunks within a certain band of the uniqueness score, chunks within a certain size band, and/or some combination of these attributes. Some clone mate statistics we can collect are whether there are any confirmed mate edges which are the result of two or more chimeric clone mate pairs, there are any chimeric clone mate pairs which participate in legitimate mate edges, and the number of unconfirmed mate edges which are real versus chimeric. We can also do some quick correctness checks based on the enhanced chunk graph such as whether the distance bounds on confirmed/unconfirmed/all mate edges are accurate.

We can make the above statistics more accurate and gather some additional statistics for unique chunks which we can extend as described above based on the partial ordering and fuzzy coordinates which result.

Limitations

Currently, the chunk graph walker will rely only on overlap edges and mate edges to traverse the chunk graph. In the future microhetergeneity within repeat chunks could also be used when other information is insufficient.

Status

Design phase.

Component Architecture and Unit Dependencies

from correctly labeled unique chunks.

AUTHORS

Granger Sutton

`Created: December 23, ’98`

`Last revised: January 22, ‘99`