Case-Based Route Planning in the Mexico City Subway System

Andrés Gómez de Silva Garza[1] and Lourdes Domenzain Ortega[1]

[1] Instituto Tecnológico Autónomo de México (ITAM), Río Hondo #1, Colonia Tizapán-San Ángel, 01000—México, D.F., México
{agomez, domen}@itam.mx

Abstract. The phrase "nearest-neighbor" is used in Case-Based Reasoning (CBR) to refer to a family of methods or heuristics for determining, during case retrieval, the similarity between a new problem to be solved and the previously-solved problems associated with the cases in a system's memory. In most problem-solving domains to which CBR has been applied, the concept of "nearness" is abstract, and determining whether a previously-solved problem is a "neighbor" to a new problem situation or not is done metaphorically. However, when the task to be performed involves route planning or related geographically-oriented activities, the original spatial connotations of "nearness" and "neighbor" come into play. In this paper we discuss the representation, storage, retrieval, and adaptation issues related to the measurement of nearness and the concept of neighborhood in case-based route planning systems in general. We also describe a particular system called SISBUSEC which plans routes in Mexico City's subway system and discuss differences and similarities with similar systems.

Keywords: case-based route planning, nearest-neighbor.

1 Introduction

The phrase "nearest-neighbor" is used in Case-Based Reasoning (CBR) to refer to a family of methods or heuristics for determining, during case retrieval, the similarity between a new problem to be solved and the previously-solved problems associated with the cases in a system's memory. In most problem-solving domains to which CBR has been applied, the concept of "nearness" is abstract, and determining whether a previously-solved problem is a "neighbor" to a new problem situation or not is done metaphorically. As an example, if the cases in a system represent the designs of buildings (*e.g.*, as in [1]) and a new set of problem requirements specified to the system include the desire to design a 30-storey office building in an earthquake zone, then the system might retrieve a case containing the design of a 35-storey office building in an earthquake zone. The values of both the *building-function* and the *ground-stability* parameters in the description of the new problem and in the retrieved case match identically, and perhaps all other earthquake zone office building cases in the case memory have a value smaller than 25 or greater than 35 for the *number-of-*

© *A. Gelbukh, Á. Kuri (Eds.)*
Advances in Artificial Intelligence and Applications
Research in Computer Science 32, 2007, pp. 269–281

Received 25/05/07
Accepted 31/08/07
Final version 30/09/07

floors parameter, so the retrieved case is the "nearest neighbor" to the new problem situation available in the case base. However, none of these relevant, descriptive features (*building-function*, *ground-stability*, and *number-of-floors*) of the problem specification and design description have spatial values. Measuring the "distance" between two values of a numeric feature (even a non-spatial one) such as *number-of-floors* is straightforward (it simply involves finding the difference between the two values), but determining "distance" between different possible values of each of the non-numeric features might have to be done in an *ad hoc* manner.

For instance, with respect to the *building-function* parameter, domain knowledge might tell us that a value of *hotel* is closer to the desired value of *office-building* than a value of *parking-lot* is, for the purpose of determining the potential usefulness of a case. This domain knowledge would have to be incorporated into the part of the case retrieval process that measures similarity. "Similarity" is a more precise term than "distance" when discussing non-spatial features, though similarity is the inverse of distance: the nearest-neighbor case would be the one that is most similar, *i.e.*, least "distant," when compared to the new problem situation. All these observations apply to many other typical problem-solving domains that have been addressed from the standpoint of CBR, such as educational, medical and legal reasoning, aside from design tasks such as the one used above for illustrative purposes.

However, when the task to be performed involves route planning or related geographically-oriented activities, the original spatial connotations of "nearness" and "neighbor" come into play. Of course, the origin of the phrase "nearest-neighbor" is spatial in origin. One well-known algorithm for tackling the NP-complete Traveling Salesperson Problem (TSP) is known as the "nearest-neighbor heuristic" precisely because it involves having the "salesperson" cyclically decide to go to the nearest city (out of all the possible cities connected to the one he/she is currently in) at each step of the algorithm until all cities have been visited [2]. Prim's algorithm for finding the minimum spanning tree of a graph (with applications to minimizing the amount of gold/copper used in printed circuit design, for instance) also proceeds in stepwise fashion by finding the node of a graph that hasn't yet been visited nearest to the node that was last visited while constructing said tree [3]. These spatial uses of the phrase "nearest-neighbor" were eventually applied to more general, more abstract, non-spatial, but analogous situations such as the determination of similarity during the generic case retrieval process.

In this paper we discuss the representation, storage, retrieval, and adaptation issues related to the measurement of nearness and the concept of neighborhood in case-based route planning systems in general. We also describe SISBUSEC, a system that we have developed which performs case-based route planning in order to find routes within a city's subway system. The system currently has knowledge about Mexico City's subway, but we will be expanding it shortly. Mexico City's subway system at present includes 11 interconnected lines that contain a total of 174 stations (counting transfer stations multiple times, once for each subway line involved in the transfer, as they are physically different and involve a displacement in space from one line to the other, even if their names are the same) and total 202 kilometers in length. As we present the relevant research issues, we also mention related work that has addressed some of these issues, sometimes in similar ways to SISBUSEC and sometimes differently.

2 Case Representation

In general a new route planning problem might be specified by stating (or, in case the problem is being solved by an autonomous robot or vehicle, by sensing) the current location and stating a desired location (presumably different from the current one). We shall use the terms "origin location" and "destination location," abbreviated l_o and l_d, to refer to these points from now on. A solution to this type of problem would be a sequence of points l_0, l_1, l_2, l_3, ..., l_n that form a route of length n between l_o and l_d, where $l_0=l_o$ and $l_n=l_d$ for the route-planning problem that said sequence is a solution to. The intermediate points l_1, l_2, l_3, ... might represent landmarks or points of interest to look out for or pass through or next to, locations or intersections where a change of route, direction, or modality of transport must be performed, etc. In a CBR system, it is this sequence of points representing the problem solution, at the very least, that must be stored in a case for reuse in future route planning scenarios.

Assuming that a system is grounded in reality, the locations l_0, l_1, l_2, l_3, ..., l_n must be described as latitude and longitude or using some other method for specifying coordinates or positions within the universe of operation of the system unambiguously. Whether it is just the labels or pointers l_0, l_1, l_2, l_3, ..., l_n that are stored in a case, and the associated spatial domain knowledge stored elsewhere, or whether the actual values of the latitude and longitude that describe the locations are stored in the case, is a design decision that must be made in a case-based route planning system.

In SISBUSEC, locations represent subway stations and a case contains just the labels of the stations that form a route from l_o to l_d. This is because subway stations are likely to be repeated in several experienced routes (each stored in its own case), especially transfer stations. If the same value for l_o is given frequently in order to plan many routes over time from the same initial location (*e.g.*, one's home or office or hotel) to many others, then those stations near to l_o will also appear in several experienced routes. Because of this, in a separate database is where we store the actual latitude and longitude (and additional knowledge, described below) related to the different subway stations in a given city, and only the names of the stations are repeated from one case to the next if several cases' solution routes involve the same stations. In the Prodigy-based system described in [4], coordinates are also used to identify specific points (within the city of Pittsburgh). On the other hand, in the Router system [5], perhaps the most similar existing system to SISBUSEC, the symbolic names used for the streets of the Georgia Institute of Technology campus and the offices, corridors, and other functional areas of the College of Computing building don't have any additional associated information to indicate their position in "the real world."

3 Case Memory Organization and Case Retrieval

In general we can classify case retrieval algorithms broadly into two different types: nearest-neighbor matching and inductive methods [6]. For a spatial task domain such as route planning, given a new problem specifying l_o and l_d, nearest-neighbor

matching would compare l_o and l_d to the endpoints l_0 and l_n of the solution route stored in a case. A perfect match occurs if $l_0=l_o$ and $l_n=l_d$, in which case the solution route stored in the case can be directly presented as a solution to the new problem. In this situation, of course, the "nearest neighbor" is not really a neighbor, but rather the exact route that connects the two locations of interest. Fig. 1 shows, both textually and graphically, this situation in which a perfect match occurs between a problem specification and a case labeled *a*. For illustrative purposes, and without loss of generality, in this and the rest of the figures in this paper we will assume that retrieved routes are of length 4 and that the graphical representation of the solution is to be read from left to right (*i.e.*, the leftmost location shown is the origin location l_o, and the rightmost location is the destination location l_d).

Situation 1: $l_0 = l_{a0}$ and $l_d = l_{a4}$
Retrieved case: *a*
Retrieved case solution: l_{a0}, l_{a1}, l_{a2}, l_{a3}, l_{a4}
Desired solution = Retrieved case solution
$= l_{a0}$, l_{a1}, l_{a2}, l_{a3}, l_{a4}

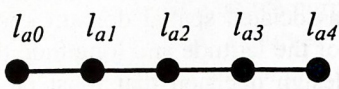

Fig. 1. Situation in which a perfect match occurs between a route-planning problem and the solution stored in a case.

A similar situation, also representing a perfect match but requiring the solution retrieved from the case to be reversed before presenting it as a solution to the new problem, occurs when $l_n=l_o$ and $l_0=l_d$. This occurs when the case contains a route from some location l_x to another location l_y and one asks a system to plan a route from l_y to l_x, and is illustrated in Fig. 2, which assumes that the match occurs between a new problem specification and a case labeled *b*. Note that this kind of situation only makes sense in route planning and perhaps a few other tasks in which reversing the components of a solution to a prior problem does not destroy the validity of the result as a solution to a new problem. On the other hand, it doesn't make sense to reverse previously-known solution routes to come up with new problem solutions in all route planning domains. In [4] reversal of routes stored in cases is not performed due to the prevalence of one-way streets in Pittsburgh. Unlike most subway lines around the world, one-way streets do not permit a traversal in both directions, so in this type of situation reversal is not a valid way to reuse the solution stored in a case.

The two situations described so far involve perfect matches, but what can a system do if there is no perfectly-matching case in memory? One solution is to give up, another is to use an alternate reasoning method instead of CBR, but a third is to retrieve the case that represents the closest-possible situation to the new problem, even if no perfectly-matching case is available. This is the type of situation in which nearest-neighbor retrieval (followed by case adaptation, which we consider below) comes into play.

Situation 2: $l_o = l_{b4}$ and $l_d = l_{b0}$
Retrieved case: b
Retrieved case solution: l_{b0}, l_{b1}, l_{b21}, l_{b31}, l_{b4}
Desired solution = Reverse (Retrieved case solution)
$= l_{b4}, l_{b3}, l_{b2}, l_{b1}, l_{b0}$

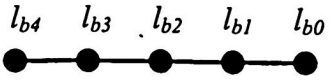

Fig. 2. Situation in which a perfect match occurs between a route-planning problem and the solution stored in a case, but the solution must be reversed before presenting it.

For route planning tasks, there are two different degrees to which one might be able to find a partially-matching case. First of all there might be a case in memory in which either l_o or l_d match one of the endpoints of the route stored in the case (but the other endpoint doesn't match—otherwise we would have a perfect match and thus one of the situations already described above). We will call this possibility P1. The second possibility, P2, is that neither l_o nor l_d match either of the endpoints of any of the solution routes in the cases stored in memory. In either situation, given a new problem specification there might be several partially-matching cases, and we would like the one that is nearest to the specification to be retrieved.

Let us assume that possibility P1 occurs, and that it's l_o that matches l_0 for a given case. If that's the only case for which l_o matches l_0, then that case is the nearest neighbor to the new problem situation and should be retrieved. However, if there are several cases for which l_o matches l_0, then some sort of tie-breaker criterion has to be implemented. The most logical tie-breaker criterion for a spatial domain such as route planning is to retrieve the case in which the other endpoint, l_n, is least distant from l_d in comparison to the other endpoints of the rest of the partially-matching cases. This case would be the nearest neighbor to the new problem situation. In the Router system, since no real positions are associated with street intersections (which is the type of location from or to which the system can plan routes), the tie-breaker criterion used is to select the first partially-matching case found. Sometimes this might turn out to be closest to the desired location than any other candidate partially-matching case, but most times it won't be.

Measuring the distance from l_n to l_d is not difficult assuming that these are not just labels but have actual positions (whether latitude and longitude, or some equivalent specification) associated with them, as in SISBUSEC (though the positions are not stored in the case, as mentioned earlier, but rather in a separate database of spatial knowledge). The nearest neighbor is the partially-matching case for which this distance is the smallest, out of all the potential candidate cases. If possibility P2 occurs, then the nearest neighbor is the case, out of all the potential candidates, for which the sum of the distances from l_o to l_0 and from l_n to l_d is the smallest.

If possibility P2 occurs it means that neither l_o nor l_d match either of the endpoints of any of the solution routes in the cases stored in memory, which means that finding

the nearest neighbor implies examining the entire case base once more (because all cases are potential candidates) after having already examined it to determine that there were no fully-matching or possibility-P1-type partially-matching cases, before deciding which case is nearest to the new problem situation. If the case base is large, this would be a time-consuming process. This is what might happen with SISBUSEC, since the Mexico City subway system has a total of 174 stations, and thus there could potentially be 174*173/2=15051 different cases (because this is the total amount of different problems that could be posed to the system) in the case memory.

This is the kind of situation that has led to the second general type of case retrieval in CBR, the inductive retrieval mechanisms. For case bases which are too large to make it feasible or logical to examine every single case to find how well it matches a new problem situation, inductive retrieval implies pre-classifying the cases into subsets. In some domains such as architectural design tasks [1] these subsets might overlap (a case may belong to several subsets), whereas in other domains it may not make sense to do so. Associated with the subsets of cases are decision-tree-like structures or procedures which provide indexing criteria by which a system can narrow down the search for a matching (or partially matching) case to a subset of the cases in memory. These structures provide an internal organization to a system's case memory, which may otherwise just be a "flat" list-like structure grouping cases together in no particular order (or perhaps in the order in which they were coded or learned, or some other arbitrary ordering that doesn't reflect the differing degrees of similarity among the cases).

In spatial reasoning domains the most natural way to group cases into subsets in order to achieve faster retrieval times is to use the concept of "neighborhood." If the geographic neighborhoods of l_o and l_d are known (or can be easily determined), and cases are somehow grouped together (classified) according to the neighborhoods that the endpoints of the routes they contain belong to, then a large number of cases can quickly be discarded without wasting time comparing them to the new problem situation during retrieval. If either a large number of neighborhoods or a large number of cases per neighborhood (both of which produce the same problem of having to make too many comparisons during retrieval, as before) result from a "flat" subdivision into neighborhoods, then another solution may be necessary. One possibility is for a hierarchical subdivision into neighborhoods and sub-neighborhoods into as many levels as needed. All of this, again, contrasts with non-spatial domains in which subsets of cases that are grouped together according to classification criteria that depend on the domain, whether the subdivision is hierarchical or not, define only abstract "neighborhoods."

In SISBUSEC, as mentioned earlier, we have a database which contains the latitude and longitude of each subway station, but it also contains knowledge about the neighborhood(s) that the stations belong to (and the connections between the stations). In order to improve the quality of the solutions produced by the system, the borders between neighborhoods in SISBUSEC are fuzzy, so stations lying on a border have more than one neighborhood associated with them. This also occurs with the Router system. In Router, however, the neighborhoods were constructed in an *ad hoc* manner, with varying numbers of neighborhoods at each level of the hierarchy and varying density of streets (or equivalent) in each neighborhood, whereas SISBUSEC

divides a geographical space into neighborhoods in a more systematic manner. The subdivision algorithm (described in the next paragraph) can therefore be easily automated in order to incorporate a new geographical area into the system without having to designate the neighborhoods of locations manually, as we did for Mexico City. The R-Finder system [7] has a two-level "hierarchy" of neighborhood-like areas referred to as "grids," into which they have divided the city of Singapore, and some streets may be contained in more than one grid in a somewhat similar fashion to SISBUSEC's and Router's use of fuzzy neighborhoods. The R-Finder "hierarchy" contains only two levels but is similar to Router's in the sense that it is a hierarchy of detail: at the highest level it is only the major roads and highways of Singapore that are represented, and at the lowest level (the grid level, where grid borders are defined by the major roads) is where the details about local roads, streets, and alleys are included. For a medium-sized city like Singapore, two levels might be enough, but if it were necessary to connect R-Finder to a similar system that contained knowledge about several Malaysian cities and the roads that interconnect them (and connect them to Singapore), or if it were necessary to group the roads in Singapore into more categories than just "major roads" and "all other roads" in order to make route planning more efficient, then more hierarchical levels may be necessary. The system described in [8] only subdivides its geographical universe of operation in the sense that it is split into cells which identify the possible initial and destination (and intermediate) locations.

SISBUSEC's method for defining neighborhoods is the following. A geographical space is divided hierarchically into quadrants (*i.e.*, into four subparts, no more and no less), producing what is known in geographical information systems [9], where it is a structure that is used frequently, and in computational geometry, as a "quad tree" [10]. Subdividing into smaller or larger amounts of neighborhoods might introduce north-south and/or east/west asymmetries into the resulting hierarchy and would not take advantage as fully of the idea of separating, at each node in the hierarchy, into a "reasonable" number of subparts. Each quadrant in SISBUSEC is recursively subdivided into quadrants if the situation merits it. This occurs when the locations in each resulting quadrant remain connected to each other within the quadrant and when the resulting quadrants end up having connections between them (otherwise the resulting "quadrants" are not really neighborhoods but very small sets of locations isolated from the rest of the locations in the geographical space). Fig. 3 shows, as a flat "map," one possible configuration of a geographical area after subdividing it in this quadrant-based fashion. Fig. 4 shows the hierarchy of neighborhoods and sub-neighborhoods that the subdivisions shown in Fig. 3 represent.

In SISBUSEC it is the leaf neighborhoods of each subway station that are stored in the database of spatial domain knowledge, since the hierarchy only serves to subdivide the geographical space. In contrast, in Router the hierarchy does not just subdivide the geographical area, but introduces more detail as one descends in the hierarchy (certain streets and therefore certain intersections are not mentioned at the root node, but appear in one or more of the neighborhoods at the next-lowest level due to their more local importance, etc.).

Returning to the issue of retrieving cases, due to the possibility that only partially-matching cases for a given new problem may be available, the potentially large size of the case base, and the spatial nature of the system's task, in SISBUSEC we have come

up with a case representation that allows us to combine aspects of nearest-neighbor and inductive retrieval. Our case memory is flat in the sense that it is just one database table (rather than several, each representing a subset of the cases) in which each row represents a given case, which would normally imply that nearest-neighbor retrieval is performed.

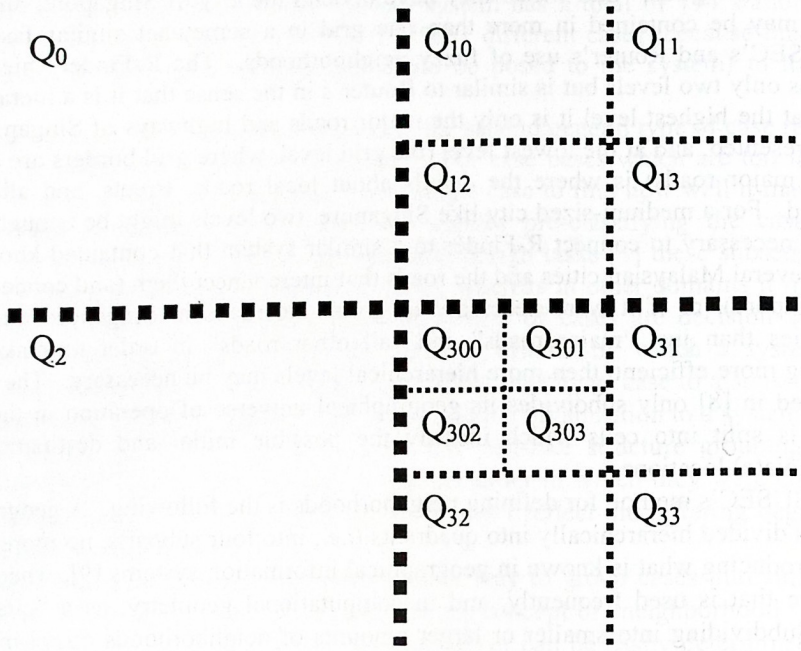

Fig. 3. Quadrant-based subdivision of a geographical space into neighborhoods and sub-neighborhoods.

On the other hand, the attributes with which we represent each case include not just the solution route, but also the origin and destination locations involved in the route, as well as their leaf-level neighborhoods, as distinct database attributes. This allows us to use l_o and l_d as primary indices during case retrieval, but also to use q_o and q_d (the quadrants or neighborhoods to which l_o and l_d belong) as secondary indices in case no perfect match occurs. The data that is retrieved if there is a match between a new problem situation and the primary and/or secondary indices of a case is the other attribute of the case: the solution (route) it contains. All of this is done without explicitly organizing the case memory into subsets of cases hierarchically. This contrasts with Router's approach, in which probing the case memory implies navigating down a hierarchical structure that reflects the organization of neighborhoods in its domain until a fully- or partially-matching case is retrieved or until a leaf neighborhood has been reached without finding a relevant case. Using q_o and q_d as secondary indices means that even if only partially-matching cases are found, they won't be just any random cases from memory, but rather they will be in the neighborhood (both literally and figuratively) of being good solutions. If no match is found between a new problem situation and any case in memory when using

even the secondary indices to probe the case memory, then it means that the system doesn't have any experience that is close enough to the new problem to be able to provide even a partially-useful solution. Case retrieval (or finding out that no cases were retrieved) in SISBUSEC is thus the result of performing between one and three quick database queries, depending on whether a perfect match, or partial matches of types P1 or P2 (as described above), occurred. Fig. 5 shows the case representation scheme used in SISBUSEC which supports this type of retrieval (each row in the database that contains the case memory describes a case in this way).

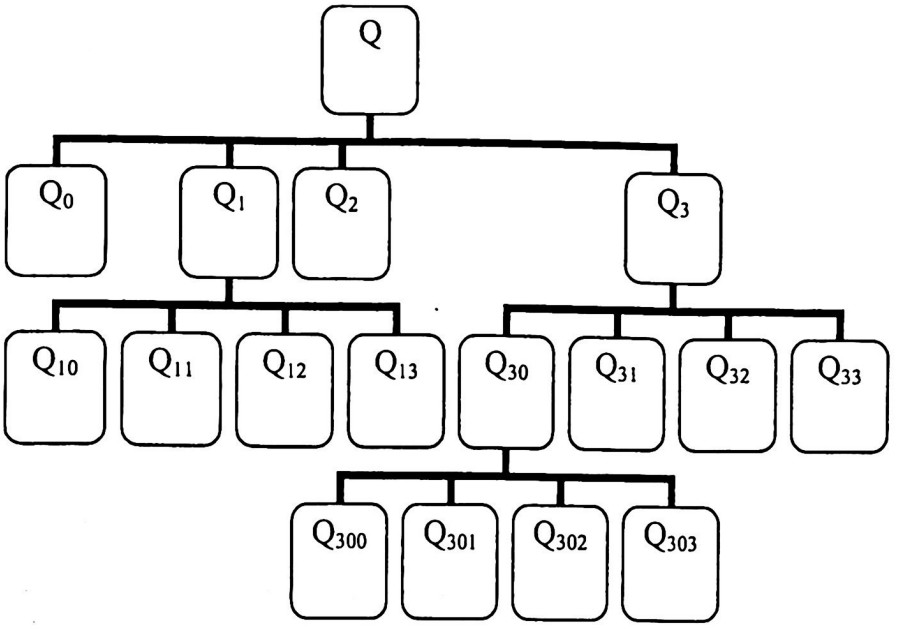

Fig. 4. Hierarchical (quad-tree) representation of the neighborhoods and sub-neighborhoods shown as a map in Fig. 3.

4 Case Adaptation

If no perfect or partial matches were found in the case memory for a given new problem, then CBR is not a feasible way to provide a solution to the problem. In these situations, SISBUSEC relies on the A* search algorithm [11], using straight-line distance as a heuristic function, to look for a solution using the graph which is implicit in the database that stores the domain knowledge about the locations and connections between the subway stations. This is similar to what happens in [4], [5], and [7], although Router's alternate algorithm is a heuristic map-based ("model-based") search that is not based on A*. If only partial matches were found during retrieval, then one of the retrieved cases must be adapted. There are several possible

situations depending on whether it was possibility P1 or P2 (described above) that occurred during retrieval. Whichever possibility occurred, what we know is that both l_o and l_d are either identical to or in the neighborhood of the endpoints of the route contained in the retrieved case. Since they are at worst close to the required locations, then case adaptation can be performed simply by extending the retrieved case (appending a new route) at one or both ends as needed. Router also operates in this fashion, as well as the system described in [8]. In SISBUSEC, the new route(s) to append are obtained using the same A* search algorithm that can be used by the system when no cases are retrieved, but are local searches (within the neighborhood(s) of the endpoints of the retrieved route) and therefore short and quick to perform.

Case *i*:

Solution: $l_{i0}, l_{i1}, l_{i2}, \ldots, l_{in}$
Primary index: l_{i0}
Primary index: l_{in}
Secondary index: q_{i0}
Secondary index: q_{in}

Fig. 5. Case representation scheme used in SISBUSEC to support neighborhood-based retrieval.

If possibility P1 occurred, then either the origin location matched one of the endpoints of the retrieved case but the destination location only matched the neighborhood of the other endpoint (possibility P1a), or else the destination location matched one of the endpoints and the origin location only matched the neighborhood of the other endpoint (possibility P1b). Fig. 6 illustrates possibility P1a assuming that the retrieved case is labeled *c*. The thick line shows the retrieved solution, and the thin line represents the appended route segment resulting from case adaptation.

Situation 3: $l_o = l_{c0}$ and $q_d = q_{c4}$ (but $l_d \neq l_{c4}$)
Retrieved case: *c*
Retrieved case solution: $l_{c0}, l_{c1}, l_{c2}, l_{c3}, l_{c4}$
Desired solution = Append (Retrieved case solution, Path(l_{c4}, l_d))
$= l_{c0}, l_{c1}, l_{c2}, l_{c3}, l_{c4}, l_d$

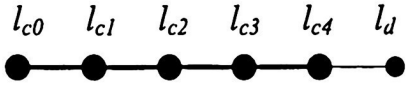

Fig. 6. Situation in which a partial match occurs between a route-planning problem and the solution stored in a case such that the origin location matches one of the endpoints of the route in the case, but the destination location is only close to the other endpoint.

A similar possibility (P1a') would occur if l_o matches the other (rightmost) endpoint, lc_4, of the retrieved case, instead of the leftmost one, lc_0, but we do not show this graphically. If possibility P1a' occurs, then case adaptation would involve reversing

the retrieved route before appending a new route segment to its right. We do not show possibility P1b graphically. A similar possibility, which we also do not show graphically (P1b'), would occur if l_d matches the other (leftmost) endpoint, l_{e0}, of the retrieved case, instead of the rightmost one, l_{e4}. This would mean that case adaptation would involve reversing the retrieved route before appending a new route segment to its left.

Assuming that neither the origin nor the destination locations were found to match any case in memory, but at least one case was found for which the neighborhoods of its endpoints match those of l_o and l_d, then possibility P2 occurred during case retrieval. This can again be broken down into two situations, P2a and P2b. In both situations, the retrieved route must be extended during case adaptation by appending new routes to both endpoints of the route obtained from the retrieved case (instead of only one endpoint, as in all the variants of possibility P1). In possibility P2a, the origin location's neighborhood matches the neighborhood of the leftmost endpoint of a case (and the destination location's neighborhood matches the neighborhood of the rightmost endpoint of the same case). In possibility P2b, the origin location's neighborhood matches the neighborhood of the rightmost endpoint of a case (and the destination location's neighborhood matches the neighborhood of the leftmost endpoint of the same case). In P2b, the retrieved route must be reversed before appending the two new routes. We do not show either of these two possibilities graphically, as they are quite similar to the situation shown in Fig. 6, except that the involve adaptation at both ends of the retrieved solution path instead of just one.

This method of adapting the route in a partially-matching case by appending one or two extra route segments to its endpoints can potentially lead to routes that contain repeated route segments. This is illustrated in Fig. 7, which assumes that the retrieved case is labeled h. In this example, it is assumed that the origin location in the new problem specification matched the leftmost endpoint of the retrieved case (*i.e.*, $lh_0=l_o$), but the destination location was only close to the rightmost endpoint (*i.e.*, $q_{h4}=q_d$, but $l_{h4}\neq l_d$). It is further assumed that the best route from l_{h4} to l_d, as found by A* during case adaptation, passes through l_{h3}, which happens to be one of the intermediate points in the solution retrieved from the case, thus ending up with the l_{h3}-l_{h4} route segment appearing twice in the route resulting from adaptation.

The Router system, which uses the same principles as SISBUSEC to adapt cases, just leaves the solutions like they were produced by the case adaptation algorithms equivalent to those described above, so some solutions would be as illogical as that shown in Fig. 7. In SISBUSEC, additionally to the above algorithms the resulting route is examined for duplicate route segments which, if found, are eliminated before presenting the route as a final solution. Thus SISBUSEC would produce the route shown in Fig. 8 as the final solution to the route that results from the first phase of adaptation as shown in Fig. 7.

In contrast to the previous approaches to case adaptation, [4] adapts cases by concatenating parts of as many cases as needed (partial cases), so the elimination of duplicate segments is not necessary (as they wouldn't appear in the constructed route—their avoidance is programmed into the case composition algorithm). A similar approach is followed by [12], though the emphasis in that system is the construction of a solution to one new problem associated with a given user from information related to several previous problems that might be stored in several case

bases, each of which belongs to (was generated by) a different user, *i.e.*, collaborative case-based route planning.

Retrieved case solution: l_{h0}, l_{h1}, l_{h2}, l_{h3}, l_{h4}
Path (l_{h4}, l_d) = l_{h4}, l_{h3}, l_d
Append (Retrieved case solution, Path (l_{h4}, l_d)) = l_{h0}, l_{h1}, l_{h2}, l_{h3}, l_{h4}, l_{h3}, l_d

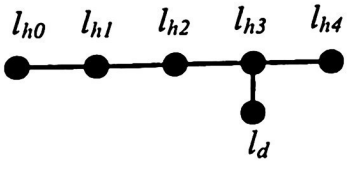

Fig. 7. Situation in which appending a route to the rightmost endpoint of a retrieved case (due to a partial match occurring during retrieval) results in a repeated route segment (and an illogical solution route).

Desired solution = RemoveDuplicateConnections (Append (Retrieved case solution, Path (l_{h4}, l_d)))
= RemoveDuplicateConnections(l_{h0}, l_{h1}, l_{h2}, l_{h3}, l_{h4}, l_{h3}, l_d)
= l_{h0}, l_{h1}, l_{h2}, l_{h3}, l_d

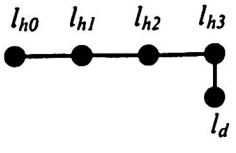

Fig. 8. Final solution route after eliminating the repeated route segments in the partial solution shown in Fig. 7 which results from case adaptation.

5 Case Learning

Since SISBUSEC can rely on the A* search algorithm to solve any problems in its problem space when a case is not available that can directly or partially help to solve the problem, then it can (and does) begin to operate even with an empty case base. This also occurs with [4], [5], [7], and [8]. When this happens, the system can store the solution as a case in order to rely more and more on these experiences, as more and more of them accumulate, when solving future problems in its domain of expertise. In SISBUSEC, this case storage/learning module takes into account the case representation and indexing scheme described above and illustrated schematically in Fig. 5.

6 Discussion

We have discussed many of the issues related to representation, storage, retrieval, and adaptation when using CBR for the task of route planning. We have also explained the decisions we have made, related to these issues, in order to implement SISBUSEC, a system that performs case-based route planning in order to plan routes within Mexico City's subway system. The name of the system is a contraction of a Spanish phrase meaning "sector-based search system" (sistema de búsqueda sectorial), from the fact that the concept of neighborhoods (quadrants, sectors) is central to the system's method for route planning. As we have shown, the concept of neighborhood has an impact in all aspects of the system's design: representation, storage, retrieval, and adaptation of cases. Finally, we have shown how some of the implementation decisions in SISBUSEC are similar and some are different from those made during the construction of similar systems that use CBR for route planning.

References

1. Gómez de Silva Garza, A., Maher, M.L.: Design by Interactive Exploration Using Memory-Based Techniques. Knowledge-Based Systems Vol. 9 No. 3 (May 1996), pp. 151-161.
2. MacGregor, J.M., Chronicle, E.P., Ormerod, T.C.: A Comparison of Heuristic and Human Performance on Open Versions of the Traveling Salesperson Problem. The Journal of Problem Solving Vol. 1 No. 1 (Fall 2006), pp. 33-43.
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (Second Edition). The MIT Press, Cambridge, Massachusetts (2001).
4. Haigh, K.Z, Veloso, M.: Route Planning by Analogy. In: Veloso, M.M. and Aamodt, A (eds.): Case-Based Reasoning Research and Development: Proceedings of the First International Conference on Case-Based Reasoning (ICCBR-95). Lecture Notes in Computer Science, Vol. 1010 (1995), pp. 169-180.
5. Goel, A.K., Ali, K., Donnellan, M., Gómez de Silva Garza, A., Callantine, T.: Multistrategy Adaptive Path Planning. IEEE Expert Vol. 9 No. 6 (December 1994), pp. 57-65
6. Wang, X.: Basic Case-Based Reasoning Techniques. Chapter 2 of A Web-Based Case-Based Reasoning Tool, Master's Thesis, University of Wyoming (2000).
7. Liu, B.: Intelligent Route Finding: Combining Knowledge, Cases and an Efficient Search Algorithm. European Conference on Artificial Intelligence (1996), pp. 380-384.
8. Kruusmaa, M., Svensson, B.: Using Case-Based Reasoning for Mobile Robot Navigation. Proceedings of the Sixth German Workshop on Case-Based Reasoning (1998).
9. Burrough, P.A., McDonnell, R.A.: Principles of Geographic Information Systems. Oxford University Press (1998).
10. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications (2nd edition, revised). Springer-Verlag (2000).
11. Hart, P.E., Nilsson, N.J., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems, Science and Cybernetics Vol. 4 No. 2 (1968), pp. 100-107.
12. McGinty, L., Smyth, B.: Identifying the Routes Best Travelled by Collaborative CBR. Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science (2001), pp. 65-74.