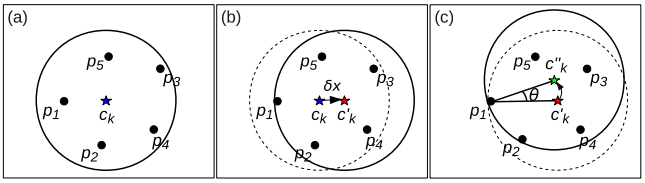

Figure 4: Finding disks to cover set of points

and rotation of the disk with center c_k. As a first step of the construction the center of the disk c_k is moved along the x axis until the first of the trajectory points inside lies on the circumference of the disk. For example in Figure 4(b) the first point which falls on the circumference after the horizontal move of the disk center is p_1. The new center of the disk is point c'_k. All points in the flock are covered by the new disk with center c'_k and diameter ϵ. Otherwise, there would be a contradiction to the assumption that p_1 is the first point on the circumference. The next step of the construction rotates the new disk using as pivot the first point on the circumference (p_1). The disk is rotated until another point falls on its circumference. In the example of Figure 4(c) the disk is rotated until point p_2 is on the circumference of disk c''_k. All points in the flock are still covered by the new disk with center c''_k and diameter ϵ (otherwise there will be a contradiction to the assumption that p_2 is the first one to be on the circumference of the disk during the rotation process). The new disk c''_k has at least two points on its circumference (points p_1 and p_2) $\square$

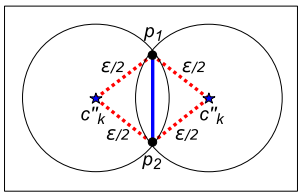

Figure 5: Disks for $\{p_1, p_2\}$, $d(p_1, p_2) \leq \epsilon$

Theorem 1 has great impact on the search for flock patterns because it limits the number of locations inside the spatial domain where to look for flocks. For a database of $|\mathcal{T}|$ trajectories there are $|\mathcal{T}|^2$ possible pairs of point combinations at any given time instance. For each such pair there are exactly two disks with radius $\epsilon/2$ that have those points on their circumference (Figure 5). We test those disks to find if they have the required minimum number of μ trajectories inside. For each time instance of the time-interval δ we have to perform $2|\mathcal{T}|^2$ tests for flock pattern. The total number of possible flock patterns that need to be tested is $2|\mathcal{T}|^{2\delta}$. In order to solve the problem, the algorithm has to not only consider each such sequence of disks (a possible flock pattern), but also to identify the trajectories that match it. The check if the trajectory stays within the sequence of disks can be done in $O(\delta)$ time. For the whole database it takes $O(|\mathcal{T}|\delta)$ time, and the total running time of the algorithm will be $O(|\mathcal{T}|^{(2\delta)}|\mathcal{T}|\delta) = O(\delta|\mathcal{T}|^{(2\delta)+1})$. As a result, the flock problem with fixed time duration has polynomial time complexity $O(\delta|\mathcal{T}|^{(2\delta)+1})$.

4. REPORTING FLOCK PATTERNS

In this section we describe a grid-based structure and some optimizations in order to efficiently compute flock disks and report flocks. We also describe five *on-line* algorithms to process spatio-temporal data in an incremental fashion.

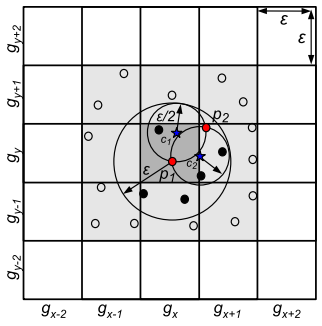

Figure 6: A grid-based index example.

The grid-based structure employed for all proposed algorithms is based on grid cells with edges of ϵ distance. Each trajectory location $p_{id}^{t_i}$ reported for a specific time instance t_i is inserted in a specific grid cell. The cell is determined by its components' location latitude and longitude. Thus, each location is inserted in only one cell. The total number of cells in the index is thus affected by the trajectory distribution in the each specific time instance t_i and the ϵ. The smaller the value of ϵ, the larger number of grid cells are needed. In our implementation, grid cells that are empty, i.e. there is no trajectory location in them, are not allocated. Other structures, e.g. *k-d-trees*, could be employed for organizing all trajectory locations in each cell grid. However, since for small ϵ the number of locations within each cell is relatively small, and given its access simplicity we used a list for each cell. The organization of this index is shown in Figure 6.

Once the grid structure is built for t_i, disks can be processed using the Algorithm 1. For each grid cell $g_{x,y}$, only the 9 adjacent grid cells, including itself, are analyzed. Algorithm 1 first process every point in $g_{x,y}$ and every point in $[g_{x-1,y-1}...g_{x+1,y+1}]$ in order to find pair of points p_r, p_s whose distances satisfy: $d(p_r, p_s) \leq \epsilon$. Because all cells in the grid index have ϵ distance, there is no need to analyze points further away of the range $[g_{x-1,y-1}...g_{x+1,y+1}]$ cells for points in a particular cell $g_{x,y}$. Pairs that have not been processed yet and are within ϵ to each other are further used to compute the two disks c_1 and c_2. In case that the pairs are exactly at distance $d(p_r, p_s) = \epsilon$, c_1 and c_2 have the same center and only one has to be further processed.

It should be noted that not all points in $[g_{x-1,y-1}...g_{x+1,y+1}]$ have to be "paired" with each point in $g_{x,y}$: only those that have distance $d(p_r, p_s) \leq \epsilon$ (as illustrated in Figure 6). Another optimization involves the points that have to be checked whether they are inside each disk computed in the previous step. Figure 7 illustrates these situations. For each point $p_r \in g_{x,y}$ (point p_1 in Figure 7(a)), a range query with radius ϵ is performed over all 9 grids $[g_{x-1,y-1}...g_{x+1,y+1}]$ to find points that can be "paired" with p_r, that is $d(p_r, p_s) \leq \epsilon$ holds. The result of such range search with more or equal points than μ ($|\mathcal{H}| \geq \mu$) is stored in the list $\mathcal{H}$ that is used to check for each disk computed. For those valid pairs, at most 2 disks are generated. For each of them, points in the list $\mathcal{H}$ are checked if they are inside the disk (Figure 7(b)). Disks that have less than μ points are further discarded and only the ones that $|c_k| \geq \mu$ holds are kept. In Figure 7(c) disk c_1 is discarded and c_2 is considered a valid disk. Because we are interested only in maximal instances of flock patterns, a