Bases de Données Spatio-Temporelles

Cédric du Mouza et Philippe Rigaux
CEDRIC/CNAM
292 rue Saint-Martin
F-75141 Paris Cedex 03
dumouza@iie.cnam.fr, rigaux@cnam.fr

26 juin 2000

Résumé

Cette article propose un état de l'art sur l'introduction de données spatio-temporelles dans un Système de Gestion de Bases de Données. Nous axons principalement la présentation sur les problèmes de gestion d'objets mobiles, en raison de l'importance de ce type d'application et de la relative simplicité du problème par rapport au spatio-temporel en général. L'article expose de manière détaillée deux aspects que nous considérons comme essentiel. En premier lieu nous consacrons un développement important à la *modélisation* des objets mobiles. Elle vise à définir un ensemble de structures et d'opérations permettant la représentation et l'interrogation par l'utilisateur. Nous traitons ensuite, plus brièvement, des techniques *d'indexation* d'objets mobiles. D'autres axes de recherche sont finalement plus brièvement survolés, et nous concluons par une bibliographie représentant les principaux travaux récents.

1 Introduction

L'avènement de technologies permettant de communiquer en temps réel avec des objets se déplaçant librement a rendu possible - et donc souhaitable - le développement de systèmes d'information gérant des données "mobiles". Cette communication est à double sens. D'un part le système GPS (*Global Positioning Systems*) permet la transmission au système central d'information précises sur la position, la direction, la vitesse des objets mobiles. D'autre part le développement des outils de communication sans fil, dont témoigne l'explosion du marché des téléphones portables [1], multiplie les échanges d'informations entre unités mobiles. A titre d'exemples, citons le suivi d'une flotte de véhicules, la fourniture aux voyageurs d'informations touristiques en temps réel, et enfin toutes les applications militaires.

Les Systèmes de Gestion de Bases de Données (SGBD) actuels ne permettent pas la représentation d'informations évoluant de manière continue. Une des caractéristiques de base dans les SGBD est que les valeurs des attributs, entre deux mise-à-jour, sont constantes, et que l'évolution de ces valeurs s'effectue de manière discrète en fonction des modifications explicitement demandées au système. A moins d'effectuer des mise-à-jour extrêmement fréquentes, on ne peut donc obtenir qu'une représentation approximative de la position d'un objet, et il est encore plus difficile de représenter les positions successives formant une trajectoire complète.

Les applications n'ont pas attendu le développement de technologies propres aux bases de données pour gérer des objets mobiles. Cependant la croissance rapide du volume des informations à traiter et la proximité de la problématique avec celles, plus traditionnelles, des bases de données spatiales [Güt94, RSV00] et des bases de données temporelles laissent à penser que les SGBD tiendront une place importante à l'avenir dans les systèmes dédiés aux objets mobiles. C'est en tout cas la conviction de nombreux chercheurs, souvent issus de l'une des deux communautés citées ci-dessus, qui sont à l'origine d'un nombre croissant de publications proposant soit l'extension de modèles et techniques initialement conçues pour les données spatiales ou temporelles, soit leur intégration.

Ces propositions visent en premier lieu à élaborer des solutions pour fournir aux applications impliquant des objets mobiles les fonctionnalités principales d'un SGBD, à savoir :

- la capacité à *représenter* l'information et à associer à cette représentation des *opérations* qui permettent d'interroger la base ;

1. L'industrie du téléphone mobile évalue à 500 millions le nombres d'utilisateurs en 2002, et 1 milliard en 2004 [Sch99].

- des *structures* d'accès aux données qui garantissent des temps de réponse acceptables même en présence de volumes très importants.

Ces aspects traditionnels sont bien représentés dans les SGBD classiques par le modèle relationnel (doté du langage SQL) et les structures de hachage ou d'arbre B qui offrent toutes des propriétés optimales en terme d'occupation de l'espace et de performance. Alors que ces questions ont étés partiellement résolues au cours des dix dernières années pour les données spatiales ou temporelles, elles restent totalement à explorer pour les données spatio-temporelles. Il faut aussi noter, à côté de ces motivations classiques, que certains aspects particuliers à ce dernier type de données, comme la gestion de l'incertitude concernant la position exacte d'un objet à un instant donné, font l'objet de recherches spécifiques.

Nous proposons dans cet article un panorama des recherches récentes consacrées aux bases de données spatio-temporelles. Bien que l'exposé qui précède puisse laisser penser que cette recherche se limite aux objets mobiles, la problématique recouvre en fait toute situation où la position et la forme d'un objet varient de manière continue au cours du temps. Cependant les problèmes de déformation tiennent une place mineure - du moins dans la littérature récente - et nous nous limiterons donc à la gestion des objets ponctuels en mouvement. Nous avons choisi de plus de présenter en détail quelques unes des principales publications récentes consacrées aux problèmes de modélisation et d'indexation, et de traiter plus brièvement des aspects que nous considérons (peut-être à tort) comme moins centraux. D'où le plan suivant pour notre présentation:

– la section 2 est un rapide rappel sur les aspects fondamentaux des bases de données, ainsi qu'une présentation des problèmes spécifiques soulevées par les données spatio-temporelles ;
– la section 3 compare trois modèles récents qui constituent chacun une approche possible pour étendre les fonctionnalités actuelles des SGBD ;
– la section 4 est consacrée à l'indexation d'objets mobiles ;
– la section 5 donne, sous forme de notes bibliographiques, un rapide aperçu sur quelques aspects complémentaires.

La bibliographie vient compléter cet état de l'art.

2 Problématique

Cette section a essentiellement pour objectif d'expliciter les problèmes posés par les données spatio-temporelles, et propose quelques principes guidant la recherche de solutions. Nous commençons par rappeler (très rapidement) les aspects principaux des SGBD, ou du moins ceux qui peuvent placer dans une perspective générale les recherches sur les BD spatio-temporelles.

2.1 Bases de données

Une base de données (BD) est un gros ensemble d'informations, le terme "gros" étant ici à interpréter relativement à la taille des informations que peut contenir la mémoire centrale d'un ordinateur. L'hypothèse de départ est que cette mémoire est beaucoup plus petite que la BD et que donc seule une faible partie des informations est disponible pour le processeur à un moment donné. Cette hypothèse a de nombreuses conséquences sur la conception du système gérant les accès à la base, le SGBD, et conduit à quelques principes de base :

1. la taille importante des données à considérer peut être compensée par la simplicité des opérations. En d'autres termes, la conception d'un langage d'interrogation doit rechercher un compromis entre le pouvoir d'expression du langage (i.e., toutes les requêtes que l'on peut exprimer) et le coût potentiel d'évaluation d'une requête. Les limitations apportées au pouvoir d'expression permettent d'assurer que les recherches demandées par l'utilisateur pourront être satisfaites par des techniques peu coûteuses ;
2. le goulot d'étranglement est constitué par l'accès aux disques (environ 100 000 fois plus long que l'accès à la mémoire vive) [GMUW00]. Il est donc crucial de disposer de chemins d'accès ou d'algorithmes permettant de diminuer le nombre de pages du disque à charger en mémoire pour effectuer une opération.

Le modèle relationnel illustre une application réussie des principes ci-dessus. Les langages relationnels, SQL en tête, offrent des possibilités relativement limitée mais, en contrepartie, leur complexité est faible [AHV95]. De plus les tables relationnelles peuvent être indexées par des arbres B qui permettent l'accès à un tuple en temps logarithmique. Ces propriétés sont fondamentales et conditionnent la viabilité d'un modèle du point de vue pratique.

Un autre aspect essentiel est la distinction de plusieurs niveaux d'abstraction dans l'architecture d'un SGBD.

1. *Le niveau physique* est celui du stockage des données. Il est entièrement (ou presque) à la charge du système qui doit assurer la sécurité, la confidentialité, et surtout la performance des accès en utilisant correctement l'espace de stockage et en définissant des chemins d'accès propres à satisfaire les requêtes les plus courantes.
2. *Le niveau logique* offre à l'utilisateur une représentation simple des données, sous forme de table par exemple dans le modèle relationnel. Cette représentation est indépendante du stockage physique des données (sur un disque ou plusieurs, en centralisé ou en réparti, etc), la transcription étant laissée à la charge du SGBD.
   La conception d'un langage de requêtes est basée sur le même souci de masquer les détails d'implantation. Idéalement, un tel langage est *déclaratif* : il permet à l'utilisateur d'exprimer ce qu'il veut obtenir, laissant au système le soin de déterminer les algorithmes à utiliser et les chemins d'accès à suivre pour récupérer les données.

Les recherches sur les modèles et les langages dans les domaines des données spatiales, temporelles et/ou spatio-temporelles visent, dans la mesure du possible, à respecter les principes rapidement survolés ci-dessus. En résumé, il s'agit de définir une représentation logique aussi simple et naturelle que possible et de s'assurer qu'il existe une transcription de cette représentation logique vers un format physique peu coûteux en espace. Le langage d'interrogation devrait être lui aussi relativement simple et naturel, et préserver un bon équilibre entre pouvoir d'expression et complexité d'évaluation. Enfin le niveau physique devrait pouvoir proposer des chemins d'accès aux données en temps logarithmique.

2.2 Données Spatio-Temporelles

Passons maintenant aux problèmes spécifiques soulevés par les données spatio-temporelles. Si on considère qu'un object spatial est caractérisé, entre autres, par un attribut géométrique décrivant sa *forme* et sa *position*, on peut définir un objet spatio-temporel comme un objet spatial dont la forme et/ou la position varient au cours du temps. En fait nous nous limitons (comme la plupart des articles) aux mouvements d'objets dont on ignore la forme (que l'on peut donc assimiler à un point). Cela revient à ignorer les zones à surface variables (incendies, déserts, marées, évolution des paysages).

On peut distinguer deux types d'applications parmi celles qui gèrent des objets mobiles. Les premières s'intéressent principalement à l'analyse des flux d'une population donnée, et considèrent des *trajectoires* décrivant l'historique, pour chaque objet, de ses déplacements successifs au cours d'un intervalle de temps qui peut être arbitrairement grand. Les requêtes envisageables sur ce type de données sont par exemple :

- donner la trajectoire de l'objet o entre t_1 et t_2 (requête temporelle) ;
- donner tous les objets dont la trajectoire passe dans la région R (requête spatiale) ;
- donner tous les objets dont la trajectoire passe dans la région R entre t_1 et t_2 (requête spatio-temporelle).

Ces requêtes requièrent une connaissance de la trajectoire complète des objets, ce qui suppose soit que l'on interroge en fait le passé (les mouvements ayant été enregistrés et stockés au fur et à mesure), soit que l'on fait des hypothèses sur la trajectoire future des objets.

Le second type d'application est beaucoup plus orienté vers le suivi (*tracking* en anglais) des objets en temps réel. On s'intéresse par exemple aux mouvements d'une flotte de taxis, des avions s'approchant d'un aéroport ou aux engins militaires sur un champ de bataille.

Dans de tels cas on cherche à satisfaire des besoins nouveaux ou à prévoir la situation dans un avenir proche plutôt qu'à faire une analyse *a posteriori* des événements. Les requêtes typiques de ce genre d'application sont [WXCJ98] :

- Trouver les taxis les plus proches du point p (un client qui attend).
- Trouver les hôtels les plus proches de l'objet o (un voyageur qui voit la nuit approcher !).
- Trouver tous les avions qui vont entrer dans la région R dans moins de 10 minutes.

C'est donc la position *courante* des objets qui est primordiale, et éventuellement la position prévisible des objets dans un futur très proche.

Les deux types d'application diffèrent sur quelques points importants. Le premier cas semble plus complexe puisqu'on ne peut se contenter de connaître la position à un instant t. Il faut conserver toute la trajectoire, ce qui implique une structure plus complexe et des données plus volumineuses. En revanche, les données étant connues complètement, on se trouve dans la situation assez confortable où la base peut être considéré comme fixe. Ce n'est pas le cas du deuxième type d'application où la position des objets est sans cesse remise en question par des mise-à-jour successives. Le problème est alors de s'assurer que l'on est capable de prendre en compte le flux des mise-à-jour qui peut être important, et de garantir que le résultat d'une requête n'est pas déjà obsolète au moment où on le transmet à l'utilisateur.

Dans les deux cas, on trouve un ensemble commun de problèmes spécifiques à la gestion de trajectoires.

Représentation

Comment représenter une valeur qui varie constamment ? La solution évidente consistant à effectuer des modifications aussi fréquentes que possibles revient à adopter une représentation discrète et est globalement insatisfaisante, aussi bien pour des raisons de précision que de surcharge du système.

Le problème est similaire à celui consistant à représenter des données géographiques linéaires (routes, rivières) : on a affaire à un ensemble infini de points que l'on ne peut représenter finiment qu'à l'aide de structures plus ou moins complexes. Le cas des trajectoires introduit un degré de complexité supplémentaire puisque ces données linéaires sont décrites non pas dans un espace à deux dimensions (x et y correspondant à la longitude et à la latitude), mais dans un espace à trois dimensions où la troisième coordonnée, le temps t, joue un rôle essentiel. Il n'y a pas de support dans les SGBD actuels pour ce type d'information.

Interrogation

Le langage traditionnel d'interrogation est SQL, de plus en plus étendu avec diverses fonctionnalités. De telles extensions ont été proposées pour des données spatiales ou temporelles, mais pas pour des données intégrant étroitement le temps et l'espace.

Indexation

Les structures traditionnelles comme l'arbre B ne sont pas adaptées aux données multidimensionnelles car elles reposent sur une structure d'ordre qui n'existe pas dans un espace dense. Des solutions ont été définies dans le cas des données spatiales qui permettent de s'approcher d'un temps de recherche logarithmique, mais ces structures (R-tree, quadtree) supposent des données statiques, et ne semblent pas adaptées à des données mobiles.

Incertitude

La notion d'incertitude est très importante à prendre en compte pour les données spatio-temporelles. La position d'un objet n'est connue qu'avec une précision relativement limitée, et surtout, sa trajectoire est soumise à diverses fluctuations qui tiennent au caractère accidenté du réseau parcouru (route) et aux variations de vitesse. Il est essentiel de tenir compte de ces marges de tolérance quand on cherche à évaluer une requête qui spécifie des critères de recherche précis.

3 Modélisation

La motivation principale des modèles présentés ci-dessous est de définir, pour un objet qui se déplace de manière continue, une représentation sur laquelle on puisse construire un ensemble d'opérations intégrables à un langage d'interrogation.

3.1 Principes généraux

Quelques idées de base sont communes à tous les modèles. En premier lieu on assimile la trajectoire d'un objet à une courbe dans un espace défini par trois variables représentant le temps t, et l'espace x et y avec l'interprétation habituelle. De plus, on ne considère le plus souvent qu'une *approximation linéaire* de la courbe, pour des raisons de simplicité de description, et également d'efficacité algoritmique.

Ensuite on se place dans un espace dense, disons $\mathbb{R}^3$, ce qui permet d'obtenir la continuité du mouvement. Dans une telle interprétation, la trajectoire d'un objet mobile est constituée d'un ensemble *infini* de points, ensemble pour lequel on doit définir une représentation *finie*. Le problème est identique dans les bases de données géographiques : un département, une route, considérés dans un espace dense, forment des ensembles infinis que l'on manipule en les représentant finiment par des structures telles que la ligne brisée formant le contour d'un polygone.

Cette approche débouche sur des outils relativement complexes pour l'utilisateur qui doit connaître les structures utilisées, maîtriser la syntaxe des opérations pour chaque structure en particulier, savoir quel est le type du résultat qui lui est fourni, etc. L'extension de cette approche à des données spatio-temporelles semble donc mener à des modèles complexes. Heureusement le caractère très particulier d'une trajectoire d'objet mobile permet de définir deux niveaux de représentation :

- **au niveau abstrait**, on peut définir un objet mobile comme une fonction continue du temps vers l'espace à deux dimensions ;
- **au niveau symbolique**, on utilise des structures qui permettent une représentation compacte de ces fonctions.

La représentation abstraite est simple à appréhender et permet de définir une interface avec l'utilisateur qui est indépendante de la représentation symbolique choisie. Cette approche fondée sur plusieurs niveaux d'abstraction est classique (voir la distinction logique/physique ci-dessus) et associe chaque acteur confronté au SGBD (l'utilisateur, le programmeur, l'administrateur) à un mode de présentation de l'information approprié.

Au niveau le plus haut on doit trouver un formalisme qui respecte la continuité de l'espace et du temps. Le modèle doit, conceptuellement, manipuler des ensembles infinis à l'aide d'un langage de requêtes déclaratif proche de SQL. A un niveau plus bas, proche de l'implémentation, le modèle propose une représentation finie des données afin de les stocker et de les manipuler.

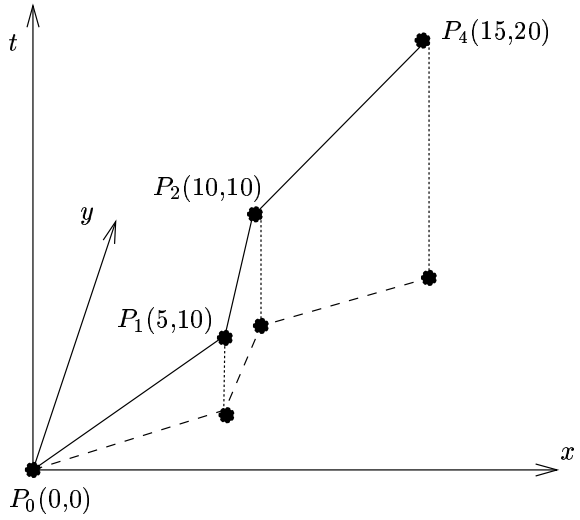


FIG. 1 – *Une trajectoire*

L'exemple de la figure 1 représente l'approximation linéaire d'une trajectoire. On peut voir, au niveau abstrait, cette trajectoire comme une fonction par morceaux qui, à chaque valeur de t sur l'intervalle $[0,5]$, associe une paire $[x,y]$.

Il existe plusieurs représentations symboliques possibles pour ces fonctions. On peut par exemple stocker les points définissant les segments, et reconstruire ensuite les fonctions par interpolation. La forme générale d'une fonction donnant une coordonnée en fonction du temps est :

$$x(t) = x_0 + v(t - t_0)$$

Donc on peut la représenter par un triplet $[x_0,t_0,v]$. Noter que la restriction à des données linéaires implique que la vitesse est supposée constante sur un segment. C'est cette forme que nous considérerons dans tout ce qui suit.

3.2 Le modèle MOST [SWCD97]

Le modèle *Moving Objects Spatio-Temporal* (MOST) a été proposé en 1997 dans [SWCD97] et implanté ensuite dans le prototype DOMINO (*Databases fOr MovINg Objects*) [WSX+99]. Le modèle gère la position présente et future des objets, mais pas l'historique de leur trajectoire, ce qui le destine principalement aux applications qui font du suivi en temps réel. L'aspect "incertitude des données" a été développé dans [WCD+98, SWCD98].

Représentation

Par opposition aux attributs statiques dont la valeur ne change que par modification discrète, les auteurs définissent les *attributs dynamiques* dont la valeur change de façon continue. Un attribut dynamique A est représenté par 3 sous-attributs :

- $A.value$: valeur de A au moment de la mise-à-jour ;
- $A.updatetime$: moment de la dernière mise-à-jour ;

– $A.function$: fonction de t qui vaut 0 à $t = 0$.

La valeur de l'attribut vaut alors, au temps $A.updatetime + t$: $A.value + A.function(t)$. L'idée est simplement qu'il est nécessaire de faire des mise-à-jour uniquement quand un objet change de direction. On stocke alors la position au moment du changement, la direction prise et la vitesse, soit les trois informations nécessaires pour représenter une valeur variant linéairement en fonction du temps.

Ce modèle permet donc de représenter de manière implicite les états futurs de la base de données (c'est à dire la position future des objets mobiles). Un *état* de la base de données est un l'ensemble des valeurs des attributs à un instant donné. Les requêtes peuvent être alors évaluées sur ces états, présents ou futurs.

Le langage FTL

Afin de tirer parti de la sémantique des attributs dynamiques, une extension de SQL par des prédicats de logique temporelle est proposée. Le nouveau langage, nommé FTL, comprend deux opérateurs temporels de base : **Until** et **Nexttime**. Etant donné deux prédicats f et g, ces opérateurs sont définis ainsi :

- f **Until** g est vérifié si on a g à cet état ou dans le futur, et f vérifié en attendant.
- **Nexttime** f est vrai si f est vérifié au prochain état de l'historique.

Nous avons dès lors une base pour construire d'autres opérateurs utilisés dans FTL :

- **Eventually** f = $true$ **Until** f.
- **Always** f = $\neg$ **Eventually** $\neg f$
- **Eventually_within_**c g : g sera satisfaite dans l'intervalle de temps c.
- **Eventually_after_**c g : g sera vérifié après l'intervalle c.
- **Always_for_**c g : g est vérifiée de manière continue pour les c prochaines unités de temps.
- g **until_within_**c h : un futur dans $[0,c]$ où h sera vérifié, en attendant g vérifié.

Ce langage permet de sélectionner les objets qui satisfont une formule construite de manière classique, dans un langage comprenant les opérateurs ci-dessus, les opérateurs arithmétiques binaires ($\leq$, $>$, $=$ ...), et quelques prédicats portant sur la partie spatiale des données.

On peut dès lors exprimer des requêtes du type "extraire tous les objets qui entrent dans le polygone P dans les 3 prochaines unités de temps, et ont l'attribut *price* $\leq$ *100"*. Le code FTL correspondant est, si l'on suppose que l'on dispose du prédicat *INSIDE(obj, area)* valant vrai si l'objet *obj* se trouve dans l'aire *area* :

retrieve *o*
where *o.PRICE* $\leq$ *100*
and **Eventually_within_3** *INSIDE(o, P)*

Evaluation

Une requête MOST, donnée sous forme de formules *conjonctives* (i.e. sans négation, car elles peuvent introduire des réponses infinies). Lors de l'évaluation d'une requête CQ spécifiée par la formule f et les variables libres $(x_1,x_2...,x_k)$, le système retourne une relation $Answer(CQ)$ avec $k + 1$ attributs : les k premiers seront une instanciation des x_i, le dernier un intervalle de validité. Par exemple si $Answer(CQ)$ est constituée des deux tuples (2, [10,15]) et (5, [12,14]), alors le système affichera l'objet qui a pour identifiant 2 entre les instants 10 et 15, et l'objet 5 entre les instants 12 et 14.

Le système proposé peut-être implémenté au-dessus d'un DBMS fournissant les fonctionnalités de base. Les auteurs ont déjà réalisé une implémentation du modèle DOMINO, selon une architecture résumée dans la figure 2.

3.3 Types abstraits spatio-temporels [FGNS00]

L'introduction de types abstraits de données (TAD) pour étendre les fonctionnalités du modèle relationnel à des valeurs complexes est maintenant relativement ancienne. L'utilisation de TAD pour la gestion des données géométriques est décrite par exemple dans [Güt89] pour la partie modélisation et [BG92] pour la partie concernant le processus d'évaluation et d'optimisation des requêtes. De nombreux systèmes sont maintenant extensibles, comme ORACLE [Sha99] ou PostgreSQL [Mom00], ce dernier proposant un ensemble complet de TAD spatiaux.

Les principes guidant la définition de TAD spatio-temporels ont été initialement présentés dans [EGSV99], puis un système de types a été proposé dans [FGNS00]. Ce modèle repose sur deux niveaux d'abstractions : le modèle "abstrait" qui permet de raisonner sur des *ensembles infinis*, sans se soucier de savoir si une représentation finie de ces ensembles existe, et le modèle dit "discret", basé sur une *représentation finie*, proche de l' implémentation. Contrairement à [SWCD97], on peut repésenter l'historique des trajectoires.

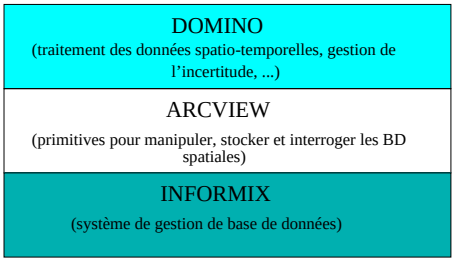


FIG. 2 – *Architecture du modèle DOMINO*

Le modèle abstrait

Le modèle se base sur un ensemble de types comprenant les *types atomiques* (*int*, *real*, *string*, *bool*, des *types spatiaux* pour représenter des données dans l'espace 2D (*point*, un point, *points*, un ensemble fini de points, *line*, une suite de courbes continues dans le plan, et *region*, ensemble fini de polygones disjoints) et enfin des *types temporels* (*instant*).

Les instances de ces types sont des valeurs statiques qui ne dépendent pas du temps. Afin de pouvoir instancier des types dont les valeurs sont dynamiques, on utilise le *constructeur de type moving*. Pour un type de données α appartenant aux types de base, il construit un *type dont les valeurs sont fonction du temps*, *moving(α)*. Une instance de ce type est un objet $\alpha(t)$. Un véhicule se déplaçant de manière continue est donc une instance de *moving(point)*.

Le modèle n'est pas limité aux objets mobiles : une région touchée par un incendie, dont la surface varie évolue en fonction du temps est du type *moving(region)*. On note plus concisément par *mpoint* et *mregion* les types *moving(point)* et *moving(region)*.

Dans le modèle abstrait, la sémantique d'une opération ou d'une fonction se définit de manière simple, du fait que l'on ne se préoccupe pas de savoir si la représentation finie existe ou non. On peut définir la syntaxe des opérateurs comme :

- **at** : *mpoint* $\times$ *time* $\rightarrow$ *point*, qui permet de connaître la position à un temps t donné.
- **mdistance** : *mpoint* $\times$ *mpoint* $\rightarrow$ *mreal*, qui renvoie un réel mobile représentant la distance (variable en fonction du temps) entre 2 points.

La sémantique est alors :

- $f_{at}(r,t) := r(t)$
- $f_{mdistance}(r,s,t) := \begin{cases} d(r(t),s(t)) & si\ r(t) \neq \perp \wedge s(t) \neq \perp \\ \perp & sinon \end{cases}$

On peut dès lors formuler des requêtes en un langage SQL étendu incorporant les opérateurs ci-dessus. Voici l'expression retournant les couples d'avions qui se croisent à moins de 500m.

select *A.id, B.id*
from *flights A, flights B*
where *A.id <>B.id*
and *minvalue$_t$(mdistance(A.route, B.route,t)) <0.5*

Le modèle discret

Il s'agit d'un niveau proche de l'implémentation, donc manipulant des ensembles finis, permettant de représenter le modèle abstrait. Tous les types du modèle abstrait ont leur équivalent dans le modèle discret. Le constructeur *moving* est remplacé par un nouveau constructeur de types.

Ainsi pour les types de base *int*, *real*, *string*, *bool*, l'implémentation se fait directement d'après les types correspondant du langage. *Point* est un couple de réel (x,y) et *points* un ensemble fini de couples (x_i,y_i). Pour *line* et *region*, on considère l'approximation linéaire (lignes brisées, polygones...). De même la représentation de *instant*, *range* et *intime* se déduit directement. Le constructeur de types ***mapping*** permet d'obtenir des types pour les objets mobiles. Un objet de type *mapping* est un ensemble $\{(I_\alpha,v)\}$, où I_α est un intervalle de temps et v la représentation de l'objet géométrique se déplaçant dans cet intervalle I_α. Concrètement I_α est l'intervalle de temps maximal pendant lequel les valeurs des attributs de cet objet

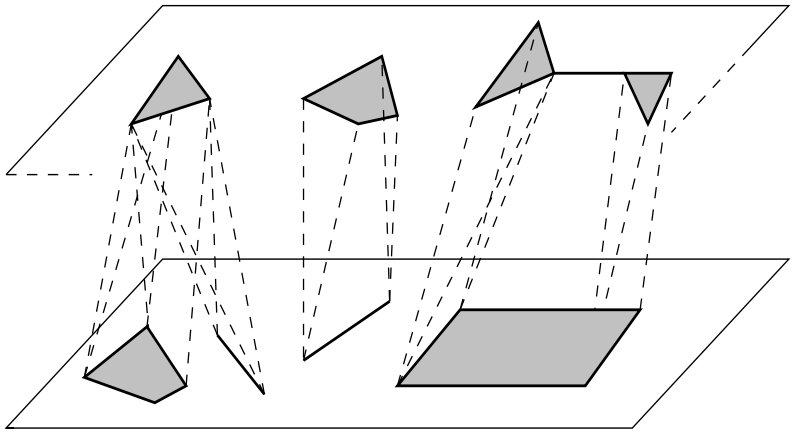


FIG. 3 – *représentation discrète d'une région mobile*

peuvent être représentées par des fonctions d'interpolation, notées ι_α. Un point mobile est donc un objet de type *mapping(upoint)* et une région mobile du type *mapping(uregion)* où :

- [**upoint**]: défini sur un intervalle de temps I_α par un quadruplet (x_0,x_1,y_0,y_1), réels représentant la position au début et à la fin de l'intervalle, tel que pour $t \in I_\alpha$ on a

$$\iota((x_0,x_1,y_0,y_1),t) = (x_0 + x_1 \times t, y_0 + y_1 \times t)$$

- [**uregion**]: basé sur un ensemble de segments mobiles qui ne se chevauchent pas. Les segments gardent la même direction pour tout t; ils ne peuvent donc faire de rotation. Ces régions peuvent avoir des trous ou non. La figure 3 page 8 montre la représentation discrète d'une région mobile.

Les définitions des fonctions dans ce modèle sont considérablement plus complexes puisque elles manipulent désormais des ensembles finis et par conséquent sont plus proche de l'implémentation. Ainsi l'opérateur at a dans ce modèle la définition suivante :
soit $r =< (p_1,t_1,b_1,c_1),...,(p_m,t_m,b_m,c_m) >$ un $mpoint$ et t un instant donné,

$$f_{at}(r,t) \quad := \quad \begin{cases} \bot & si\ m = 0 \vee (m > 0 \wedge (t < t_1 \vee t > t_m)) \\ p_i & si\ m \geq 1 \wedge (\exists i \in \{1,...,m\} : t_i = t) \\ lin(p_i,t_i, & \\ \quad p_{i+1},t_{i+1},t) & si\ m \geq 2 \wedge (\exists i \in \{1,...,m-1\} : (t_i < t < t_{i+1}) \wedge b_i \wedge \neg c_i \\ p_i & si\ m \geq 2 \wedge (\exists i \in \{1,...,m-1\} : (t_i < t < t_{i+1}\} \wedge b_i \wedge c_i \\ \bot & si\ m \geq 2 \wedge (\exists i \in \{1,...,m-1\} : (t_i < t < t_{i+1}\} \wedge \neg b_i \end{cases}$$

où $lin(p_1,t_1,p_2,t_2,t)$ est une fonction qui retourne le point p déduit de l'interpolation linéaire entre les 2 points (p_1,t_1) et (p_2,t_2) au temps t. Le drapeau b_i vaut vrai si le point est défini entre t_i et t_{i+1}. Le drapeau c_i vaut faux si une interpolation linéaire entre p_i et p_{i+1} peut être utilisée.

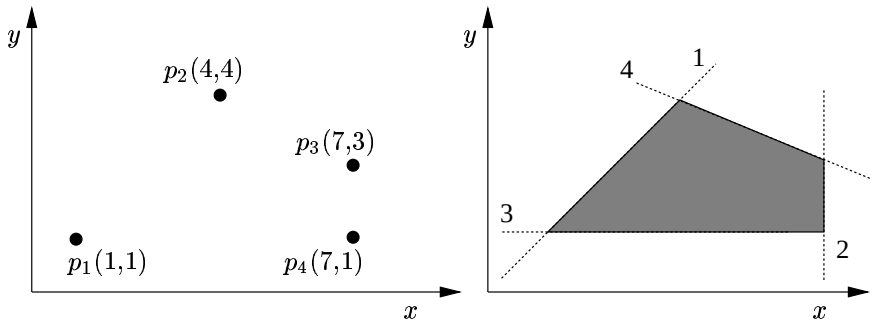
3.4 Modèle contraintes [GRS00]

Le troisième modèle que nous présentons s'appuie sur les bases de données contraintes qui proposent un cadre formel permettant de raisonner sur les données géométriques. Nous introduisons brièvement l'intuition avant de développer l'application aux données spatio-temporelles.

Bases de données contraintes

Le modèle de données contraintes a été proposé initialement en 1990 par Kanellakis, Kuper et Revesz [KKR95]. L'idée de base est qu'il est beaucoup plus facile d'interroger une base de données multidimensionnelle en considérant que la base est constituée de *l'ensemble* des points, plutôt que de raisonner sur la représentation finie de cet ensemble de points qui peut reposer sur des structures relativement complexes.

Essentiellement, le modèle contraintes propose une extension du modèle relationnel consistant à exprimer des requêtes avec un langage relationnel classique (disons la logique du premier ordre) sur des relations



Un ensemble fini de points Un ensemble infini de points

FIG. 4 – *Idée de base : ensemble de points = relation*

$Spat$	
x	y
1	1
7	3
4	4
7	1

Une instance finie

$Spat$	
x	y
1	1
1.001..	1
1.002..	1
1.003..	1

$Spat$ (cont.)	
x	y
...	...
3	2
3.001..	2
...	...

Une instance infinie

FIG. 5 – *Instances de la relation $Spat$*

infinies dont chaque tuple est un des points d'un objet géométrique (voir Fig. 4. Un polygone par exemple est un ensemble infini de points, considéré comme une relation sur deux attributs, x et y (Fig. 5).

Le fait de raisonner sur des ensembles de points (en fait sur des relations) simplifie considérablement la tâche de l'utilisateur. Une opération de *clipping* par exemple, sélectionnant la partie d'un polygone $Spat$ située dans un rectangle $Rect$, s'exprime par une simple jointure.

select *x, y*
from *Spat, Rect*
where *Spat.x = Rect.x*
and *Spat.y = Rect.y*

Bien entendu le raisonnement doit s'appuyer sur une représentation finie, soit en l'occurence des *contraintes*. Ce niveau, dit *niveau symbolique* - est caché à l'utilisateur mais constitue un support pour l'évaluation des requêtes.

La représentation avec contraintes peut être vue comme une extension de la représentation - finie - relationnelle, au prix d'un léger changement de point de vue. Au lieu de considérer une relation comme un ensemble de tuples, on doit maintenant la voir comme une formule de logique du premier ordre. Par exemple un ensemble fini de quatre points est représenté par la formule :

$$\phi_{Spat} \equiv (x = 1 \wedge y = 1) \vee (x = 7 \wedge y = 3) \vee (x = 4 \wedge y = 4) \vee (x = 7 \wedge y = 1)$$

Le langage logique utilisé dans ce cas ne comprend que le symbole d'égalité, et aucun symbole d'opération arithmétique. En étendant le langage, on obtient la possibilité de représenter un ensemble infini, comme par exemple le polygone de la figure 4.

$$\begin{aligned} & y \leq x && (1)\\ \wedge \quad & x \leq 7 && (2)\\ \wedge \quad & y \geq 1 && (3)\\ \wedge \quad & x + 3y - 16 \leq 0 && (4) \end{aligned}$$

La sémantique de cette formule est définie naturellement comme l'ensemble des paires de valeurs pour (x,y) dans le domaine des réels qui la rendent vraie (dans le domaine des réels). On peut interpréter géométriquement cette représentation comme l'intersection d'un ensemble de demi-plans, chacun étant représenté par une inéquation sur x et y.

Il est possible de représenter, avec des contraintes linéaires utilisant seulement l'addition, toutes les figures géométriques courantes (polygones, lignes, points). De plus *il n'y a a pas de limite sur la dimension des objets représentés.* La dimension de l'espace dans lequel sont décrits les objets géométriques correspond en fait au nombre de variables utilisées : 2 pour un espace 2D, 3 pou un espace 3D, etc. Un polytope (convexe) en dimension trois P est décrit par une conjonction d'équation d'hyper-plans sur des variables x,y et z.

$$\begin{array}{c} 32x + 5y - 34z \leq 98 \\ \wedge\ 40x - 28y + 3z \leq 30 \ldots \end{array}$$

L'évaluation de requêtes relationnelles sur des données géométriques consiste alors à déterminer quelle est la formule qui représente le résultat de la requête. Par exemple la sélection relationnelle $\sigma_{2x-3y+8z\leq 3}(P)$ donne le résultat suivant.

$$\begin{array}{c} \mathbf{2}\ x\ \mathbf{-}\ \mathbf{3}\ y\ \mathbf{+}\ \mathbf{8}\ \mathbf{z} \leq \mathbf{3} \\ \wedge\ 32x + 5y - 34z \leq 98 \\ \wedge\ 40x - 28y + 3z \leq 30 \ldots \end{array}$$

On a simplement effectué la conjonction de la formule représentant P et de l'argument de la sélection ! Bien entendu il reste à évaluer l'ensemble de points représentés par cette nouvelle formule, ce qui met en oeuvre une algorithmique sur les contraintes linéaires. Le livre [KLP00] permet d'en savoir plus sur tous les aspects très rapidement survolés ci-dessus.

Application aux objets mobiles

Ce modèle a été développé par le CNAM et l'INRIA et présenté dans [GRS00]. L'idée est plus générale que l'application aux objets mobiles : on s'intéresse en fait à des classes d'objets géométriques placés dans un espace de dimension d mais dont la dimension "intrinsèque" est inférieure. Cette propriété est caractérisée dans le modèle par l'existence de *fonctions d'interpolation* permettant d'obtenir une des variables décrivant un objet comme une fonction d'autres variables.

Représentation

Deux interpolations très courantes sont l'interpolation linéaire à l'aide de 3 points, comme celle utilisée dans le TIN (*Triangulated Irregular Networks*) et l'interpolation linéaire à l'aide de 2 points comme celle d'une trajectoire. Le principe du TIN est de découper l'espace en triangles, en stockant à chaque sommet son "altitude". Pour un point $P(x,y,h)$ quelconque, on fait une interpolation linéaire avec les 3 sommets $P_i(x_i,y_i,h_i)$:

$$h = \varphi_1(x,y) \times h_1 + \varphi_2(x,y) \times h_2 + \varphi_3(x,y) \times h_3$$

où les φ_i sont des coefficients dépendant de la distance de P aux sommets P_i.

Le modèle contraintes permet une formalisation simple de cette classe des objets mobiles. Nous avons vu qu'une trajectoire est approximée par une suite de segment connectés deux à deux dans un espace de dimension 3. La trajectoire de la Fig. 1 peut être représenté en mode contraints par la formule :

$$\begin{array}{ll} x = 10t\ \wedge\ y = 5t\ \wedge\ 0 \leq t \leq 1 & \text{(a)} \\ \vee & \\ y = 5t\ \wedge\ x = 10\ \wedge\ 5 \leq t \leq 10 & \text{(b)} \\ \vee & \\ 3x = 10t + 10\ \wedge\ 3y = 5t + 20\ \wedge 2 \leq t \leq 5 & \text{(c)} \end{array}$$

A l'instant $t = 0$, l'objet mobile est situé au point P_0. Puis il va de P_0 à P_3, où il arrive à $t = 5$, en passant par P_1 ($t = 1$) et P_2 ($t = 2$). Les variables x et y dont des fonctions (linéaires) du temps t, ce qui correspond à la constatation informelle qu'une trajectoire est une ligne (de dimension intrinsèque 1) décrite dans un espace de dimension 3. Une définition générale de ce type de données consiste à les considérer comme des relations **infinies** dotées des propriétés suivantes :

- Elles ont une **clé**, par exemple le temps t dans le cas des objets mobiles.
- Il y a donc une **dépendance fonctionnelle** de la clé vers les autres attributs, soit $t \rightarrow x$ et $t \rightarrow y$.

La représentation avec contraintes correspond alors à un type particulier de formules :

- Des contraintes quelconques (avec $\leq$, $\geq$) sur les attributs de la clé (ici t).
- Pour chaque autre attribut v, une égalité $v = f(clé)$.

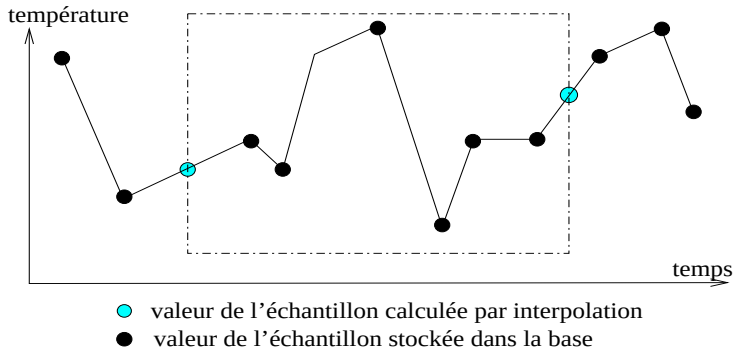


FIG. 6 – *exemple du calcul de l'échantillon à partir des valeurs stockées et de la fonction d'interpolation*

Ceci définit une forme normale pour la représentation d'objets mobiles. Une trajectoire est représentée symboliquement par des contraintes linéaires de la manière suivante :

$$Traj(x,y,h) = \bigvee_i \; t_i(t) \wedge x = f_i(t) \wedge y = g_i(t)$$

où les $t_i(t)$ sont les contraintes définissant l'intervalle de temps, et les f_i,g_i sont les fonctions linéaires donnant les coordonnées.

Cette représentation débouche sur un modèle de données possédant les propriétés suivantes.

- une représentation abstraite *indépendante* du stockage physique
- un langage de requête *simple*
- une complexité *raisonnable* de l'évaluation des requêtes

Un des principes de ce modèle est de **cacher les données échantillon et les fonctions d'interpolation aux utilisateurs**. Ces données et ces fonctions doivent être liées étroitement. De plus les données échantillon doivent être calculées par le système et non définies par l'utilisateur (exemple d'un fenêtrage, figure 6 page 11).

Le fait que ni les échantillons, ni les fonctions, ne soient visibles par l'utilisateur garantit :

- l'**exactitude** puisque l'utilisateur ne peut utiliser une fonction d'interpolation inadaptée, ou des échantillons non valables.
- la **convivialité** car l'utilisateur dispose d'un langage de requêtes standard et n'a pas à se soucier de l'interpolation.
- la **sûreté** puisqu'ici la réponse à une requête peut être infinie.

Evaluation

Si les requêtes sont exprimées sur des relations abstraites (infinies), l'évaluation doit se faire sur un modèle fini. Un des résultats obtenus sur ce modèle est que l'évaluation peut-être réduite à un petit nombre d'opérateurs appliqués aux attributs qui forment la clé de la relation. Ainsi les opération de sélection et de jointure, par exemple, se réaliseront dans le sous-espace formé par les attributs-clés. Les algorithmes développés par les auteurs permettent d'interroger les données stockées (donc représentation finie) à partir d'une requête du modèle abstrait. Ils exploitent la forme interpolée des données et organisent l'évaluation comme une combinaison des opérations "primitives" que sont la *projection*, l'*intersection*, l'*image d'une fonction*, le *rectangle englobant minimal*.

Soit par exemple l'objet suivant, où z est une fonction de la clé (x,y).

$$3x - 5y \leq 2 \;\wedge\; 5x + 45y \leq -8 \;\wedge\; 3x - 9y \leq 34$$
$$\wedge\; z = 4x + 3y - 1$$

La requête $\sigma_{2x-3y+8z\leq 3}(P)$ donne pour résultat, en effectuant une simple conjonction des formules :

$$\mathbf{2}\,x\,\mathbf{-}\,\mathbf{3}\,y\,\mathbf{+}\,\mathbf{8}\,\mathbf{z} \leq \mathbf{3}$$
$$\wedge\; 3x - 5y \leq 2 \;\wedge\; 5x + 45y \leq -8 \;\wedge\; 3x - 9y \leq 34$$
$$\wedge\; z = 4x + 3y - 1$$

Il est facile de voir qu'il suffit de substituer $4x + 3y - 1$ partout où apparaît z pour obtenir la formule équivalente :

$$\begin{array}{c} \mathbf{2}\ x\ \mathbf{-}\ \mathbf{3}\ y\ \mathbf{+}\ \mathbf{8}\ \mathbf{(4}\ \mathbf{x}\ \mathbf{+}\ \mathbf{3y}\ \mathbf{-}\ \mathbf{1)} \leq \mathbf{3} \\ \wedge\ 3x - 5y \leq 2\ \wedge\ 5x + 45y \leq -8\ \wedge\ 3x - 9y \leq 34 \\ \wedge\ z = 4x + 3 \end{array}$$

L'interprétation de cette opération est donnée dans la figure 7. On peut ramener tout calcul sur l'ensemble des variables à un calcul sur l'espace défini par (x,y). En d'autres termes, une algorithmique en dimension 2 est suffisante.

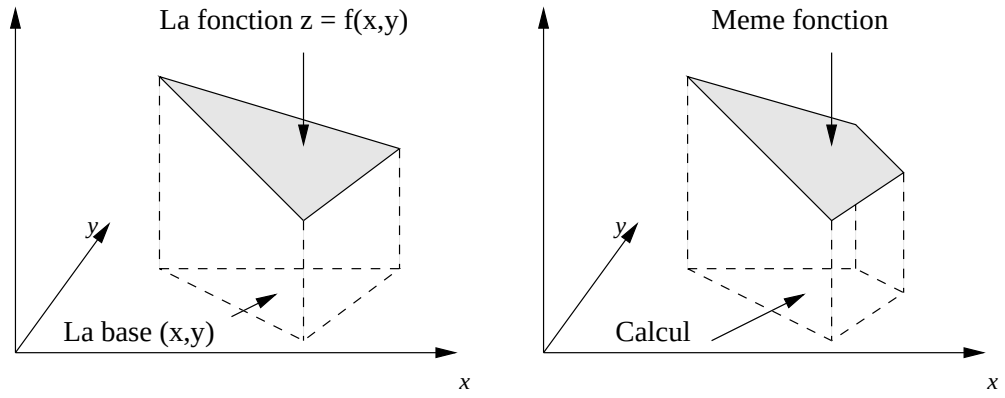


FIG. 7 – *Technique d'évaluation par substitution de variables*

Interrogation

Pour l'utilisateur, il n'y a que des relations, et un langage comme SQL. Voici quelques exemples, basés sur le schéma suivant :

- Une carte $Map(x,y,libellé)$ qui donne l'occupation du sol en chaque point (x,y).
- Un MNT $MNT(x,y,z)$ qui donne l'altitude en chaque point.
- La trajectoire d'avions, $Traj(t,x,y,a)$, avec la position et l'altitude à chaque instant.

Les requêtes suivantes illustrent la simplicité de cette approche. Il n'y a pas besoin de penser à la machinerie interne (comme : les structures, les opérations, la syntaxe, le type de mon résultat, etc). Exprimer une requête se limite à considérer les données comme des ensembles de points ou, de manière équivalente, comme des relations. Que ces relations soient finies ou infinies n'a plus d'importance puisque nous avons vu qu'il existe un format fini et des opérations sur ce format qui permettent l'évaluation.

1) Donner la position et l'altitude de l'avion au temps $t1$.

```
select  x, y, a
from    Traj
where   t = 't1'
```

2) Donner la trajectoire de l'avion avec son altitude par rapport au sol.

```
select  t, t1.x, t1.y, a - h
from    TIN t1, Traj t2
where   t1.x = t2.x and t1.y = t2.y
```

3) Montrer les forêts entre 1 000 et 2 000 mètres.

```
select  t.x, t.y
from    TIN t, Map m
where   t.x = m.x and t.y = m.y
and     1000 ≤ h ≤ 2000
and     name = 'forest'
```

4) Montrer les parties de la trajectoire où l'avion était au-dessu de la mer, à une altitude supérieure à 1000 m.

```
select  l.x, l.y, l.t
from    Traj l, Map m
where   m.x = l.x and m.y = l.y
and     name = 'sea'
and     a > 10000
```

4 Stockage et indexation

Cette section est consacrée à un deuxième problème fondamental, l indexation d'objets mobiles. Nous commençons par un rappel des deux techniques les plus couramment utilisées pour indexer des données multi-dimensionnelles, à savoir le R-tree et la quadtree linéaire, avant d'exposer deux adaptations récentes destinées aux objets mobiles.

4.1 Indexation de données multi-dimensionnelles

Il existe une multitude de techniques d'indexation d'objets multidimensionels [GG98], dont la plupart sont des variantes de l'une des deux catégories suivantes :

- **Partitionement du jeu de données** en fonction de leur répartition dans l'espace. Essentiellement, on cherche à regrouper les objets proches dans l'espace dans les mêmes pages du disque.
- **Découpage régulier de l'espace**. Dans ce cas on ne tient pas compte de la distribution du jeu de données à indexer. L'espace de référence est découpé *a priori* en cellules, régulières ou non. Les objets sont ensuite affectés aux cellules avec lesquelles ils ont une intersection.

La principale structure de la première catégorie est le R-tree, dont un exemple est donné dans le figure 8. Le R-tree est construit sur une hiérarchie de rectangles contenus les uns dans les autres, le niveau le plus bas étant constitué des rectangles minimaux englobant la géométrie des objets spatiaux. A chaque niveau, les rectangles sont groupés en "paquets" pouvant être stockés sur une page du disque.

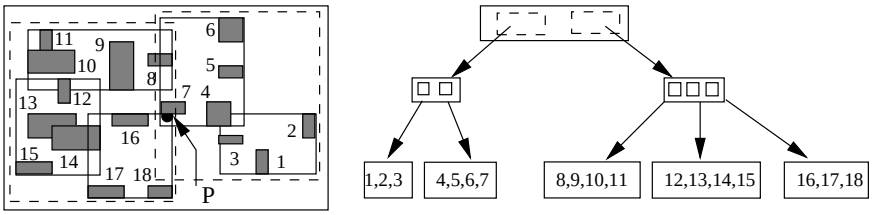


FIG. 8 – *Indexation par R-tree*

Dans l'exemple de la figure 8, on suppose qu'une page peut stocker au plus 4 objets, ce qui donne les groupes {1, 2, 3}, {4, 5, 6, 7}, etc. Le regroupement des objets est basé sur leur proximité dans l'espace. On cherche en fait à minimiser le recouvrement des rectangles englobant de chaque groupe (les rectangles dessinés en traits fins sur la figure). Ces rectangles sont à leur tour regroupés pour former le niveau supérieur, et ainsi de suite jusqu'à ce que l'on obtienne un dernier groupe de moins de 4 éléments, stockés sur une page qui forme la racine de l'arbre.

Le R-tree a des propriétés comparables à celles de l'arbre-B : l'arbre est équilibré, sa hauteur est logarithmique dans la taille du jeu de données, et sa complexité en espace est linéaire. La recherche (pointé par exemple) basée sur un R-tree consiste à parcourir, à chaque nievau (en partant de la racine) le sous-arbre dont le rectangle englobant contient le point argument. Malheureusement le R-tree ne garantit pas un temps de recherche logarithmique. La recherche des objets contenant le point P par exemple devra parcourir 5 pages (la racine, les deux noeuds de niveau 1, et deux feuilles), sans ramener un seul objet. Un autre inconvénient de la structure est le coût de l'algorithme qui divise les objets d'un noeud quand celui-ci est trop plein.

La deuxième structure largement utilisée (dans ORACLE par exemple) sous une variante ou une autre est le *quadtree*. Dans sa variante la plus simple, on découpe l'espace en un ensemble de cellules régulières et on associe une page à chaque cellule. Chaque objet est alors inséré dans toutes les cellules qu'il intersecte, ce qui peut mener à référencer plusieurs fois un même objet.

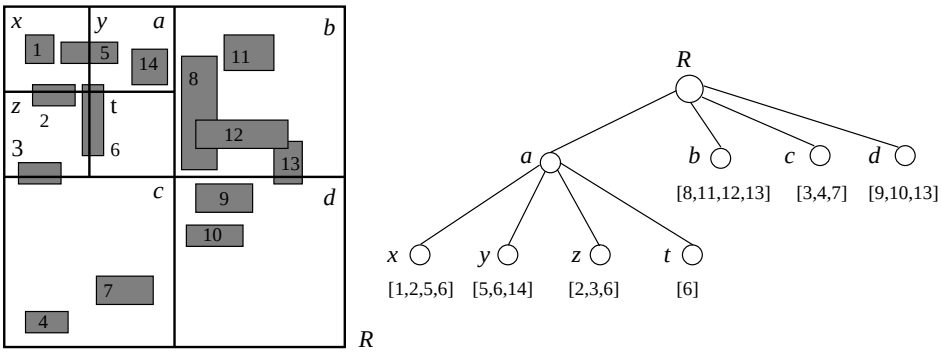


FIG. 9 – *Indexation par quadtree*

Quand une page déborde, on la divise en quatre correspondant à quatre sous-cellules égales, et on répartit les objets entre ces quatre pages/cellules. Un exemple est donné dans la figure 9, en supposant qu'une page contient 4 objets. Le coin supérieur gauche est plus divisé puisqu'il contient 5 objets.

Cette méthode est très simple, et comprend de nombreuses variantes intéressantes puisqu'elles permettent de réutiliser l'arbre B du SGBD. L'inconvénient est la redondance qui impose un tri pour éliminer les doublons après chaque requête.

En résumé il n'existe pas de structure optimale pour indexer des objets dans un espace de dimension 2. Les solutions ci-dessus apparaissent cependant comme des bons compromis qui se rapprochent du comportement souhaité (temps de recherche logarithmique, occupation de l'espace linéaire) dans le cas de données ayant des propriétés statistiques acceptables (et notamment une réprtition dans l'espace à peu près uniforme). Si on considère maintenant que les objets sont mobiles, aucune des deux structures ci-dessus ne semble tenir la route. Dans l'un et l'autre cas, l'indexation est basée sur l'hypothèse que la position des objets est fixe.

4.2 Indexation par Quadtree [TUW98]

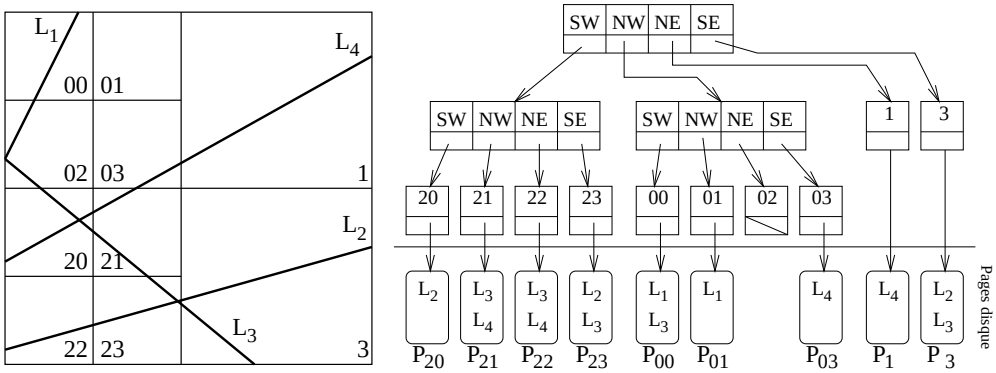
Comme le précédent, cet index vise à indexer les positions future, et se rapproche donc du modèle [SWCD97] auquel il se réfère d'ailleurs explicitement. Chaque trajectoire est donc un segment de droite commençant à t_0 et s'étendant sur un intervalle de temps prédéfini. Elle est représentée par deux fonctions linéaires $x = v_x \times t + x_0$ et $y = v_x \times t + x_0$.

4.2.1 Construction

Les trajectoires sont indexées par une variante du quadtree dite *PMR-quadtree*. Afin de se ramener au cas où on indexe des segments dans un espace 2D, on utilise en fait deux quadtrees : un pour la fonction donnant $x(t)$ et l'autre pour $y(t)$. Lors d'une recherche, il faut utiliser séparément les deux index, puis effectuer l'intersection des résultats obtenus.

La construction de l'index est basée sur le découpage récursif classique du quadtree. Etant donné un segment s décrit par (v_x,x_0), on recherche tous les quadrants qui intersectent s et on y insère l'information (id,v_x,x_0) où id est l'identifiant permettant de rechercher l'objet sur le disque. Comme toutes les structures basée sur un découpage régulier de l'espace, le même objet est référencé plusieurs fois (redondance).

Lorsqu'il y a un dépassement de capacité de la page d'indexation, le quadrant est divisé en 4 régions (qu'on appelera suivant leur position géographique NW, SW, NE et NW). L'enregistrement $< id,v_x,x_0 >$ est ajouté tous les nouveaux quadrants traversés. Les figures 10 montre un exemple de l'indexation proposée par les auteurs. Dans cette exemple la capacité de stockage d'une page est fixée à 2. L_1, L_2, L_3 et L_4 sont 4 trajectoires d'objets mobiles traversant l'espace d'indexation. Par exemple le quart en bas à gauche de notre espace est traversé par 3 trajectoires (L_1, L_2 et L_3). Il a donc fallu le découper en quatre quadrants et associer autant de pages du disque (P_0, P_1, P_2 et P_3).



a. Les trajectoires b. Le quadtree

FIG. 10 – *Construction du quadtree*

Mise-à-jour

Il s'agit d'une des principales limites de la structure : que faire quand un objet change de direction ? La solution proposée par les auteurs est assez brutale : un index est construit pour une période de temps déterminée ΔT et devient obsolète à l'issue de cette période. Il faut alors reconstruire complètement un nouvel index pour la période suivante, et ainsi de suite.

Recherche

L'index supporte les fenêtrages spatio-temporels : étant donnée les intervalles $[x_{min},x_{max},y_{min},y_{max},t_{min},t_{max}]$, trouver les objets qui vont passer dans cette fenêtre (NB : l'intervalle temporel doit être compris dans l'intervalle de validité de l'index). On effectue deux fenêtrages sur les deux quadtree, le premier avec le rectangle $[x_{min},x_{max},t_{min},t_{max}]$, le second avec $[y_{min},y_{max},t_{min},t_{max}]$, l'algorithme consistant à parcourir récursivement, en partant de la racine, les quadrants intersectant la fenêtre.

Résultats

La première remarque faite par les auteurs concerne la taille du stockage de cette indexation. Le nombre de pages du disque nécessaire croit en fonction du nombre d'objets par paliers. La valeur d'un palier est 4 fois plus grande que celle du palier précédent. Cela est du au fait que les quadrants en cas de dépassement de capacité sont coupés en 4 et que dans les résultats présentés on a supposé une équi-répartition des données (par conséquent le dépassement de capacité à lieu pour tous les quadrants à peu près au même moment).

Les tests effectués pas les auteurs permettent également d'évaluer le taux de remplissage de l'arbre quadtree. Ce taux est une fonction du nombre d'objets que l'on souhaite indexer. La valeur de celui-ci est comprise, d'après les auteurs, entre 50% et 90%. Cette valeur de 90% s'explique par le fait que la i^{eme} "vague" de segmentation des quadrants, correspondant au i^{eme} palier cité au paragraphe précédent, commence avant que la $(i-1)^{eme}$ soit finie.

4.3 Indexation par R-tree : le TPR-tree [SJLL00]

Le TPR-tree (*Time-Parameterized R-tree*) est une extension du R-tree pour indexer les positions actuelles et futures d'objets mobiles, et non les historiques des positions des objets mobiles. L'index s'applique donc à un modèle de données comme celui de [SWCD97], et pas à ceux qui permettent d'interroger la trajectoire passée. Chaque objet est décrit une position de référence $x^i(t_{ref})$,$v^i(t_{ref})$, i variant entre 1 et la dimension de l'espace considéré. La position sur l'axe i à un instant $t > t_{ref}$ est obtenue par $x^i(t) = x^i(t_{ref}) + v^i(t_{ref} \times (t - t_{ref}))$.

Il y a également 3 paramètres qui affectent l'indexation et ses performances. Le premier est l'intervalle de temps dans le futur W qu'une requête peut interroger, intervalle ayant pour borne inférieure l'instant présent. Le deuxième est l'intervalle de temps U pendant laquelle on suppose que l'index peut-être utilisé, qui a donc pour borne inférieure l'instant de création de l'index. Enfin le dernier est l'intervalle H qui contient tous les temps spécifiés dans les requêtes (il vaut en fait $W + U$).

Le temps correspondant à la position de référence d'un objet mobile t_{ref} est choisi égal au temps de chargement de l'index. De plus la période de validité d'un index qui doit être fixée, afin d'avoir une indexation pertinente, en fonction des caractéristiques des objets étudiés.

Construction

Dans un arbre R, chaque rectangle à n'importe quel niveau de l'arbre englobe un ensemble d'objets. Cette propriété est essentielle pour garantir la possibilité d'effectuer correctement une recherche. L'idée de base de l'index TPR est que les rectangles doivent évoluer avec le temps de manière à préserver cette propriété à tout instant.

La figure 11 illustre l'intuition. On considère 6 objets mobiles, et on suppose qu'une page peut contenir au plus trois objets. La partie haute représente les positions à l'instant T1, et la partie basse les mêmes objets à l'instant T2 (T2 > T1).

Clairement un rectangle fixe est obsolète à T2 (partie gauche de la figure), et ne peut plus servir à la recherche. Si on prend maintenant des rectangles mobiles, le choix des groupes d'objets ne doit plus seulement tenir compte de la proximité dans l'espace, mais aussi de leur position future. Par exemple (partie centrale de la figure 11) grouper les objets d'après leur proximité donne un bon résultat pour le groupe $\{1,2,3\}$, mais pas pour $\{4,5\}$ et $\{3,6\}$ puisque on sein de chaque groupe les directions sont divergentes, ce qui aboutit à de très larges rectangles au temps T2. En revanche, un regroupement qui tient compte des deux paramètres (partie droite de la figure 11) donne de bons résultats.

Pour des raisons de coûts élevés de stockage, on ne peut avoir à tout instant le rectangle englobant minimal. Les auteurs proposent un rectangle englobant qui est minimal pour $t = t_{ref}$ et qui croit suivant les vitesses maximales des objets qu'il contient. Ce qui signifie que la taille d'un rectangle croît toujours, et ne constitue pas le rectangle minimal des objets à tout instant. La figure 12 donne un exemple de l'évolution du rectangle englobant entre l'instant t_{ref} et un instant $t > t_{ref}$. A l'instant de référence t_{ref} , qui correspond donc à l'instant où est chargé l'index, les différents objets a, b, c, d, e, f et g ont une position et une vitesse donnée. On notera $(x_a(ref),y_a(ref))$ et $(v_{a,x}(ref),v_{a,y}(ref))$ la position et la vitesse de l'objet a à l'instant t_ref. Ainsi à un instant $t > t_ref$ l'objet a a pour position $(x_a(ref) + v_{a,x}(ref).(t - t_{ref}),y_a(ref) + v_{a,y}(ref).(t - t_{ref}))$.

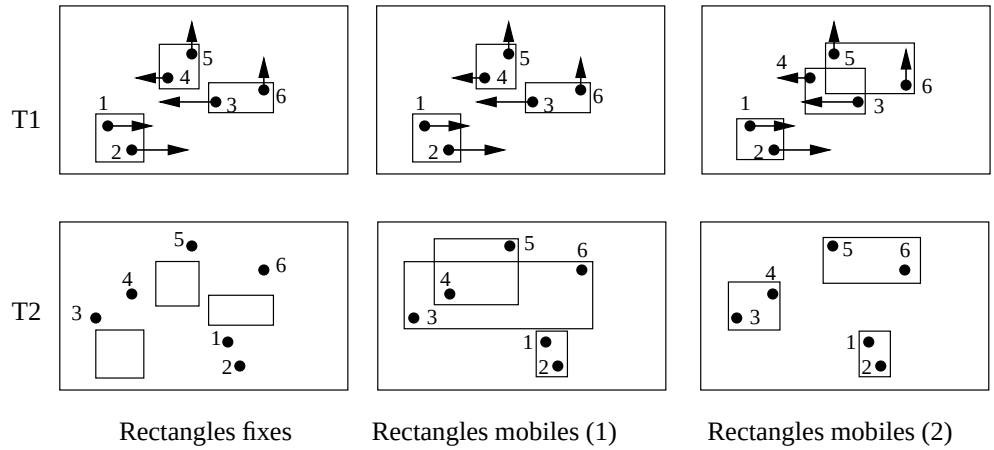


FIG. 11 – *Rectangles mobiles dans le TPR-tree*

La vitesse de croissance du rectangle englobant est obtenue quant à elle en cherchant les vitesses minimales et maximales en abscisse et ordonnées, en valeurs algébriques, des objets qu'il contient. Ainsi par exemple pour les abscisses, la vitesse du bord droit du rectangle se déplacera à la vitesse correspondant à la valeur maximale des différentes composantes horizontales des vitesses des objets contenus, soit dans notre cas $v_{a,x}(t_ref)$, et la vitesse du bord gauche à la valeur minimale, soit ici $v_{b,x}(t_ref)$. L'exemple de la figure 12 illustre bien par ailleurs le caractère non minimal du rectangle englobant pour un instant $t > t_{ref}$.

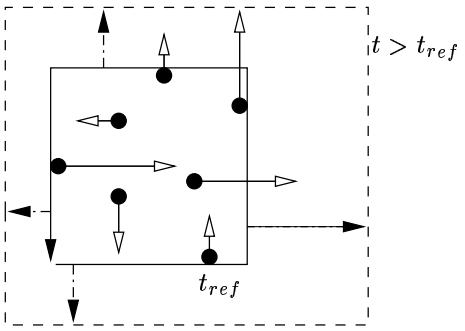


FIG. 12 – *Croissance d'un rectangle mobile*

Les algorithmes d'insertion et suppresion d'objets dans un R-tree classique visent à minimiser certains paramèters qui conditionnent la qualité de la structure obtenue. Ces paramètres sont la surface occupée par les rectangle, le recouvrement des rectangles, le périmètre (qui indique si le rectangle est plus ou moins proche d'un carré). Dans le cas du TPR-tree qui utilise des rectangle variant avec le temps, on doit cherche à minimiser les paramètres ci-dessus pour l'ensemble de l'intervalle de temps t_{ref},$t_{ref} + H$ pendant lequel l'arbre va être utilisé. On va donc utiliser l'intégrale de la fonction donnant la valeur de ce paramètre. Par exemple on cherchera à minimiser l'expression suivant, donnnant la surface d'un rectangle au cours de son déplacement :

$$\int_{t_1}^{t_1+H} Aire(t).dt$$

La généralisation des algorithmes du R-tree (en fait d'une de ses variantes, le R*tree [BKSS90]) en remplaçant les paramètres habituels par une intégrale est relativement directe. Nous renvoyons à l'article [SJLL00] pour plus de détails.

Mise-à-jour

Là encore une des principales limites de la structure est la mise-à-jour de l'index.La solution proposée par les auteurs est différente de la précédente: l'index créé est valable en théorie pour un temps infini.

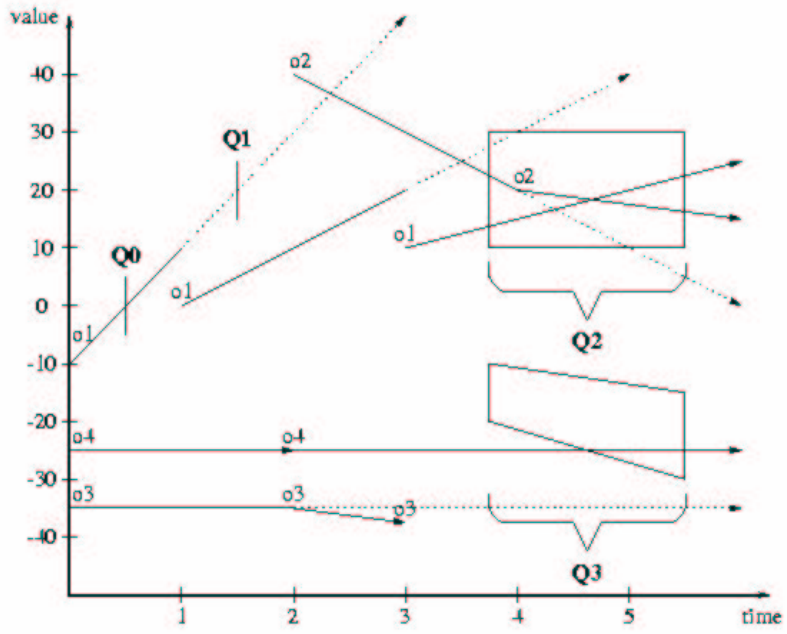


FIG. 13 – *exemples de requêtes dans un espace d'une dimension*

Néanmoins ses performances se détériorent avec le temps. Cet index a un intervalle de validité: il est construit pour une période de temps déterminée où il est optimisé, fixé par le paramètre U. Au delà de cette période il est préférable pour l'exactitude des résultats de reconstruire l'index.

Recherche

L'index proposé par les auteurs doit permettre de résoudre trois types de requêtes:

1. les requêtes correspondant à un fenêtrage spatial R à un temps t donné.
2. les requêtes correspondant à un fenêtrage spatial R lors d'un intervalle de temps $[t_{deb},t_{fin}]$.
3. les requêtes correspondant à un fenêtrage mobile, valant R_{deb} à t_{deb} et R_{fin} à t_{fin}.

La figure 13 page 17 montre des exemples de ces 3 requêtes dans un espace à une dimension. Dans cet exemple, oX désigne un objet mobile et sa trajectoire et on suppose que l'on réalise une mise-à-jour toutes les 1 unité de temps. $Q0 = ([-5,5],O.5)$ est une requête du premier type qui a pour réponse l'objet $o1$. $Q1 = ([15,25],1.5)$ est également une requête de type 1, cependant 2 cas se présentent: soit la requête a été formulée avant $t = 1$, c'est à dire avant la mise-à-jour, auquel cas la réponse retournée est l'objet $o1$ pour lequel on a prédit la trajectoire future , sinon la réponse est l'ensemble vide puisque la position et la vitesse de l'objet $o1$ ont déjà été mises à jour. La requête $Q2 = ([10,30],3.75,5.5)$ est une requête de type 2. Si la requête a été effectuée à un temps $t < 1$, la réponse est l'ensemble vide puisque la trajectoire prévue pour $o1$ ne traverse pas la fenêtre de la requête, et que l'existance de l'objet $o2$ n'est pas encore connue. Si la requête est réalisée à $1 < t < 2$ la réponse est $o1$ puisqu'après la mise-à-jour, sa trajectoire prévue intersecte la fenêtre de la requête. L'existance de $o2$ n'est toujours pas connue. Par contre si la requête a été réalisée à un instant $t > 2$ alors la réponse retournée est $o1$ et $o2$. Pour $Q4$, quel que soit l'instant t où l'on effectue la requête, la réponse retournée est $o4$.

Voyons maintenant comment sont évalués les différents types de requêtes et notamment comment s'effectue la recherche dans le TPR-tree. Dans le cas d'une requête (R,t_0), donc de type 1, la recherche s'effectue de manière assez semblable au R-tree classique: on sélectionne un rectangle englobant $mbb(t)$ si à l'instant t_0 où la requête est évaluée, la fenêtre de la requête R intersecte le rectangle englobant à l'instant t_0 $mbb(t_0)$. Ainsi en dimension 1 par exemple, si $R = (a_{min},a_{max})$ et le rectangle englobant est défini par $(x_{min},x_{max},v_{min},v_{max})$, on sélectionne ce rectangle englobant si $a_{min} \leq x_{max} + v_{max}(t_0 - t_{ref}) \wedge a_{max} \geq x_{min} + v_{min}(t_0 - t_{ref})$

Pour répondre aux requêtes de types 2 et 3, définies sur un intervalle de temps $[t_{deb},t_{fin}]$, il faut sélectionner les rectangles englobants qui intersectent la fenêtre de requête entre t_{deb} et t_{fin}. Pour les requêtes de type 3, les auteurs proposent un algorithme qui s'appuie sur l'observation que 2 rectangles mobiles s'intersectent si il y a un instant t, $t_{deb} \leq t \leq t_{fin}$, où leurs projections dans chacune des dimensions s'intersectent. Ainsi l'algorithme présenté va calculer pour chaque dimension l'intervalle de temps pour lequel les deux rectangles s'intersectent. Si l'intersection de tous ces intervalles est nulle, alors les deux rectangles n'intersectent pas. Sinon, l'intervalle résultat correspond à l'intervalle de temps pendant lequel les deux rectangles intersectent.

Résultats observés

Le premier résultat observé par les auteurs est lié à l'importance de l'intervalle H (cf section précédente). Les meilleurs résultats sont obtenus pour une valeur proche de l'intervalle moyen des mises à jour des objets mobiles.

Ils remarquent également que plus les données contenues dans un rectangle englobant ont un comportement proche, plus l'indexation avec un R-tree ou mieux encore un TPR-tree est performante au niveau du nombre d'entrées-sorties. En effet le rectangle englobant aura une expansion au cours du temps limité et restera proche du rectangle englobant minimal.

Les performances de l'indexation proposée dépendent également de la taille de l'intervalle temporel qui correspond au futur du moment où est exprimé la requête, que l'on peut interroger. Plus cet intervalle augmente, moins les performances sont bonnes. Néanmoins elles restent supérieures à celle d'un R-tree classique.

Enfin les auteurs notent que le nombre d'opérations d'entrées-sorties ne croit que très lentement en fonction du nombre d'objets représentés. De plus si les performances se dégradent au début avec le temps, elles stagnent ensuite car l'arbre se stabilisera.

5 Notes bibliographiques

La gestion de données spatio-temporelles est en train de devenir un axe important de recherche dans la communauté bases de données, comme en témoignent l'apparition de *workshops* entièrement dédiés à la question [JS99, Spa99], ainsi que la création de groupes de travail comme le réseau eruropéen Chorochronos [FGG+99, KS00].

La position des objets est connue avec une relative incrtitude, ce qui soulève des problèmes très spécifiques à ce type d'application. Une adaptation du langage FTL pour prendre en charge les requêtes est décrite dans [WCD+98]. Le même article discute des politiques de mise-à-jour des attriutes dynamiques prenant en compte l'incertitude. Les auteurs de [PJ99] considèrent eux le traitement de l'incertitude des positions fournies par le système GPS, ainsi que l'impact sur les techniques de recherche. Le problème a aussi été étudié dans le cadre des bases de données contraintes [Kou97].

Les techniques d'indexation n'ont été étudiées que récemment [TSPM98]. Outre les travaux cités, [KGT99] propose des structures optimales en espace et en temps pour l'indexation de la position future des objets mobiles. Les techniques reposent sur la transformation d'une équation $x = v(t-t_0)+x_0$ (v étant la vitesse et x_0 la position initiale) en un point (v,x_0). Ce point peut alors être indexé dans un espace "dual". La viabilité des structures dépend de beaucoup de restrictions qui rendent problématiques leur application pratique. La même remarque vaut pour [AAE00], la proposition que nous avons détaillée ([SJLL00]) nous paraissant la plus convaincante à ce jour. L'indexation des trajectoires (incluant l'historique des positions) a fait également l'objet de quelques travaux, consistant essentiellement à adapter la construction de l'arbre R [TUW98, NST99, PTJ99].

La recherche sur les objets mobiles se heurte à la difficulté d'obtenir des jeux de données variés et volumineux. Les études de performannce notamment ont besoin d'évaluer les techniques proposées sur des données présentant des profils statistiques bien identifiés, d'où la spécification de générateurs de données dans [SM99, TSN99], tous deux disponibles sur le WEB. Quelques articles relativement isolés reprennent des problématiques issues bases de données spatiales pour les généraliser au spatio-temporel : langages spatio-temporels [BTAPL99], ...

Références

[AAE00] P.K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proc. ACM Symp. on Principles of Database Systems*, pages 175–186, 2000.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BG92] L. Becker and R.H. Güeting. Rule-Based Optimization and Query Processing in an Extensible Geometric Database System. *ACM Transactions on Database Systems*, 17(2):247–303, 1992.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*tree : An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 322–331, 1990.

[BTAPL99] C. Bonhomme, C. Trépied, M.-A. Aufaure-Portier, and R. Laurini. A Visual Language for Querying Spatio-Temporal Databases. In *Proc. Intl. Symp. on Geographic Information Systems*, pages 34–39, 1999.

[EGSV99] M. Erwig, Güting, M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3):269–296, 1999.

[FGG+99] A.U. Frank, S. Grumbach, R. H. Güting, C.S. Jensen, M. Koubarakis, N.A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, T.K. Sellis, B. Theodoulidis, and P. Widmayer. Chorochronos: A Research Network for Spatiotemporal Database Systems. *SIGMOD Record*, 28(3):12–21, 1999.

[FGNS00] L. Forlizzi, R.H. Gueting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2000.

[GG98] V. Gaede and O. Guenther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2), 1998.

[GMUW00] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, 2000.

[GRS00] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating Interpolated Data is Easier than you Thought. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2000.

[Güt89] R.H. Güting. Gral: An Extensible Relational Database System for Geometric Applications. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 1989.

[Güt94] R.H. Güting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4):357–399, 1994.

[JS99] C. Jensen and M. Scholl, editors. *Proc. of the VLDB workshop on Spatio-Temporal Database Management*, Edinburgh (Scotland), September 1999.

[KGT99] G. Kollios, D. Gunopolos, and V.J. Tsotras. Nearest Neighbor Queries in a Mobile Environment. In *Intl. Workshop on Spatio-Temporal Database Management (STDBM'99)*, LNCS 1678, 1999.

[KKR95] P. Kanellakis, G Kuper, and P. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995. A shorter version appeared in PODS'90.

[KLP00] G. Kuper, L. Libkin, and J. Paradaens, editors. *Constraint Databases*. Springer Verlag, 2000.

[Kou97] M. Koubarakis. The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases. *Theoretical Computer Science*, 171(1/2), 1997.

[KS00] M. Koubarakis and T. Sellis, editors. *Spatiotemporal Databases: The Chorochronos Approach*. Springer Verlag, 2000. http://www.dbnet.ece.ntua.gr/˜choros/.

[Mom00] Bruce Momjian. *PostgreSQL, Introduction and Concepts*. Addison Wesley, 2000. To appear. See `http://postgresql.org`.

[NST99] M.A. Nascimento, J.R.O. Silva, and Y. Theodoridis. Evaluation of Access Structures for Discretely Moving Points. In *Intl. Workshop on Spatio-Temporal Database Management (STDBM'99)*, LNCS 1678, 1999.

[PJ99] D. Pfoser and C.S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, pages 111–132, 1999.

[PTJ99] D. Pfoser, Y. Theodoridis, and C. S. Jensen. Indexing Trajectories of Moving Point Objects. Technical report, Chorochronos Network, 1999.

[RSV00] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases*. Morgan Kaufmann, 2000.

[Sch99] A. Schieder. Wireless Multimedia Communications: An Ericsson View. In *Information Multimedia Communications Symposium*, 1999.

[Sha99] J. Sharma. Oracle8i Spatial: Experiences with Extensible Databases. An Oracle Technical White Paper, May 1999.

[SJLL00] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2000.

[SM99] J.-M. Saglio and J. Moreira. Oporto: A Realistic Scenario Generator for Moving Objects. In *Proc. Intl. Conf. on Databases and Expert System Applications (DEXA)*, pages 426–432, 1999.

[Spa99] S. Spacapietra, editor. *Proc. of the DEXA Workshop on spatio-temporal data models and languages*, Firenze (Italy), September 1999. IEEE Computer Society.

[SWCD97] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 422–433, 1997.

[SWCD98] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the Uncertain Position of Moving Objects. In *Temporal Databases: Research and Practice*, volume 1399, pages 310–337. Springer-Verlag, 1998.

[TSN99] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the Generation of Spatiotemporal Datasets. In *Intl. Conf. on Large Spatial Databases (SSD'99)*, 1999.

[TSPM98] Y. Theodoridis, T. K. Sellis, A. Papadopoulos, and Y. Manolopoulos. Specifications for Efficient Indexing in Spatiotemporal Databases. In *Intl. Conf. on Scientific and Statistical Database Management*, 1998.

[TUW98] J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree Based Dynamic Attribute Indexing Method. *Computer Journal*, 41:185–200, 1998.

[WCD+98] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 588–596, 1998.

[WSX+99] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. DOMINO: Databases fOr MovINg Objects tracking. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 547–549, 1999. (Demo sessions).

[WXCJ98] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*, pages 111–122, 1998.