|  |  |  |  |
| --- | --- | --- | --- |
| **S Université Constantine 2 –Abdelhamid Mehri**  **Faculté des nouvelles technologies**  **Département du IFA/ Master 1 RSD** | | | |
|  |  |  |

Pro1

Rapport 1

Base de Donnée Distribuée Orienté Graphe

09/04/2019

**Préparé par : KARIMOU SEYNI Ibrahim**

Table des matières

[Approche de la Théorie de Graphes et de Jeux 1](#_Toc2157926)

[1. Notions mathématiques 2](#_Toc2157927)

[2. Optimisation combinatoire 3](#_Toc2157928)

[3. Théorie de Graphe 5](#_Toc2157929)

[4. Théorie de jeux 7](#_Toc2157930)

[5. Partitionnement de graphe 9](#_Toc2157931)

[Apprentissage Automatique 9](#_Toc2157932)

[6. Histoire 10](#_Toc2157933)

[7. Principe 11](#_Toc2157934)

[8. Type d’apprentissage 11](#_Toc2157935)

[9. Exemple 13](#_Toc2157936)

[Base de Donnée Distribuée Orienté Graphe 14](#_Toc2157937)

[Historique 15](#_Toc2157938)

[Motivation 16](#_Toc2157939)

[Architecture 16](#_Toc2157940)

[Logique temporelle arborescente (CTL) & Langage de requete 17](#_Toc2157941)

[1. Logique temporelle arborescente (CTL) 17](#_Toc2157942)

[2. Langage de Requête 20](#_Toc2157943)

[Conclusion 26](#_Toc2157944)

[1. Principe Modèle Checker Distribué 28](#_Toc2157945)

[2. Paramètre d’optimisation 29](#_Toc2157946)

[3. Principe d’équilibre de Nash 30](#_Toc2157947)

[4. Principe de Optimisation (α, β, γ) 30](#_Toc2157948)

[5. Algorithme de distribution d’espace d’états basée sur le comportement des systèmes 44](#_Toc2157949)

[1. Génération de l’espace d’état distribue 45](#_Toc2157950)

[2. Construction des statistiques et des rapports 45](#_Toc2157951)

[3. Politique de redistribution 46](#_Toc2157952)

[Tableau 1 Propriétés CTL 19](#_Toc525461453)

[Tableau 2 Expresions reguliers neo4j 22](#_Toc525461454)

[Tableau 3 Séries d'exemple de requêtes neo4j 26](#_Toc525461455)

[Figure 1 Modélisation par graphe, des sept ponts de la ville de Königsberg 15](#_Toc525461470)

[Figure 2 Graphes reparties dans différentes machines 17](file:///E:\Master1%20RSD\Memoire\raport\BDDOG.docx#_Toc525461471)

[Figure 3 Vue global des graphes distribue 17](file:///E:\Master1%20RSD\Memoire\raport\BDDOG.docx#_Toc525461472)

[Figure 4 La structure d'un graphe 20](file:///E:\Master1%20RSD\Memoire\raport\BDDOG.docx#_Toc525461473)

# Approche de la Théorie de Graphes et de Jeux

Depuis le problème des ponts de Königsberg ***[*EULER*, 1736],*** la théorie des graphes s'est particulièrement développée en raison du nombre élevé de problèmes qu'elle permet de résoudre ces dernières décennies, La modélisation mathématique facilite la compréhension d’un problème, car elle détermine un seul vocabulaire formel pour différentes situations, et elle permet de trouver une méthode de résolution efficacement et optimale. La théorie des graphes a une très large gamme d’applications dans divers domaines, en particulier chez les mathématiciens et les ingénieurs **[Deo, 2017].**

La théorie de jeux n’en fait pas exception qui focalise sur les jeux de société et acteur formalise des situations sociales et donc des processus d'interactions entre des individus sous la forme de jeux. Le recoure a cette théorie est d’aboutir à une bonne stratégie face à des problèmes de type économique puis militaire et politique **[J&O, 1944].**

En informatique, les graphes jouent un rôle important dans de nombreuses branches ; ils sont utilisés pour représenter les interactions statiques ou dynamiques dans des réseaux complexes (interactions entre amis dans un réseau social ou son évolution, l’activation des gènes dans les systèmes biologiques, des réactions chimiques dans les réseaux biochimiques). En général, un graphe peut être utilisé pour représenter toute situation impliquant des objets discrets et des relations entre eux ; les graphes sont ensuite analysés pour découvrir les propriétés de l’objet modélisé, ou transformés pour construire d’autres types de modèles.

De jour en jour, des données sont générée, les problèmes devient de plus en plus complexe à traiter, la taille des graphes des problèmes augmente constamment. Beaucoup de graphes sont maintenant trop gros et dépasse même les limites de la mémoire. Pour pouvoir charger le graphe en mémoire et l’utiliser efficacement, le graphe doit être partitionné. Cependant, le partitionnement peut être fait sous certain contrainte, les stratégies de la théorie de jeux permettent d’aboutir à des meilleures partitions respectant les différentes contraintes.

Nous présenterons au cours de ce chapitre quelques rappels de mathématiques, de la théorie des graphes, la théorie de jeux. Ils seront suivis d’une description formelle du partitionnement de graphe et de ses objectifs. Puis les deux problèmes de partitionnement seront présentés : le partitionnement contraint et le partitionnement non contraint. Une avant-dernière section étudiera la NPdifficulté de ces problèmes de partitionnement. La dernière section classera les différents problèmes de partitionnement que nous étudierons en fonction de leur caractère contraint ou non contraint.

## Notions mathématiques

[BICHOT, 2007]

**Définition 1** (*Ensemble*)

Un ensemble S = (s1, . . ., sn) désigne une collection d’éléments. Les ensembles utilisés dans ce document sont finis et leur taille est notée |S|. On note ∅ l’ensemble vide.

**Définition 2** (*Cardinal d’un ensemble*)

Soit X un ensemble fini d’éléments. Le cardinal de l’ensemble X est égal au nombre d’éléments de X, et on le note **card**(X).

**Définition 3** (*Partition*)

Soit un ensemble S quelconque. Un ensemble P de sous-ensembles de S est appelé une partition de S si :

* Aucun élément de P n’est vide ;
* L’union des éléments de P est égale à S ;
* Les éléments de P sont deux à deux disjoints.

Les éléments de P sont appelés les parties de la partition P. Le cardinal de la partition P est alors le nombre de parties de P.

Le partitionnement de graphe fait partie des problèmes d’optimisation combinatoire, qui est une branche des mathématiques discrètes. Ce dernier parfois appelées mathématiques finies, sont l’étude des structures mathématiques où la notion de continuité est absente. Les objets étudiés en mathématiques discrètes (tels que les entier relatifs, les graphes simples et les énoncés en logique [Norman]) sont des ensembles dénombrables. Dans le cadre des mathématiques discrètes, un ensemble dénombrable (ensembles qui ont la même cardinalité que les sous-ensembles des nombres naturels, y compris les nombres rationnels mais pas les nombres réels) peut aussi être appelé ensemble discret.

## Optimisation combinatoire

L’optimisation combinatoire aussi appelée optimisation discrète, est une branche de l'optimisation en mathématiques appliquées et en informatique, également liée à la recherche opérationnelle, l'algorithmique et la théorie de la complexité, elle consiste à trouver un "meilleur" choix parmi un ensemble fini (souvent très grand) de possibilités. Elle recouvre les méthodes qui servent à déterminer l'optimum ou montrer la difficulté de résoudre une fonction sous des contraintes données, contrairement aux fonctions sans contrainte la solution optimal correspond au coût optimal (minimal, maximal) de la fonction.

La plupart des problèmes d’optimisation appartiennent à la classe des problèmes **NP-difficile** classe où il n’existe pas d’algorithme qui fournit la solution optimale en temps polynomial en fonction de la taille du problème et le nombre d’objectifs à optimiser. D’où la nécessité d’utiliser les méthodes approcher (Méta heuristique, Heuristique, etc.) pour obtenir l’ensemble de solution admissible aux problèmes.

Dans ce qui suit nous présentons les concepts et vocabulaires lié au domaine.

**Définition 1 (**Fonction objectif)

Une fonction objectif est une fonction qui modélise le but à atteindre Dans le problème d'optimisation sur l'ensemble des critères. Il s'agit de la fonction qui doit être optimisée. Elle est notée F(x) de manière générale F(x) est un vecteur :

F(x)= [f1(x), f2(x), …, fk(x)]. Elle est aussi appelée : critère d’optimisation, fonction coût, fonction d’adaptation, ou encore performance.

**Définition 2 (**Paramètres)

Un paramètre du problème d’optimisation, est une variable qui exprime une donnée quantitative ou qualitative sur une dimension du problème: coût, temps, taux d’erreurs, etc. Ces paramètres correspondent aux variables de la fonction objective. Ils sont ajustés pendant le processus d’optimisation, pour obtenir les solutions optimales. On les appelle aussi variables d’optimisation, variables de conception ou de projet.

**Définition 3 (**Vecteur de décision)

Un vecteur de décision est un vecteur correspondant à l'ensemble des variables du problème, il est noté : = avec : n le nombre de variables ou dimension du problème et xk la variable sur la dimension K

**Définition 4 (**Critère de décision)

C’est un critère sur lequel sont jugés les vecteurs de décision pour déterminer le meilleur vecteur. Un critère peut être une variable du problème ou une combinaison de variables.

**Définition 5 (**Contraintes)

Une contrainte du problème est une condition que doivent respecter les vecteurs de décision du problème. Une contrainte est notée : gi () avec i=1,…, q. q : le nombre des contraintes

**Définition** 5 (Solution admissible).

Une solution admissible est un ensemble de valeurs données aux variables qui satisfait toutes les contraintes.

**Définition 6 (**Espace de recherche)

Représentant l’ensemble des valeurs pouvant être prises par les variables.

**Définition 7 (**Espace réalisable)

**Définition 8** (Solution optimale)

Une solution optimale est une solution admissible qui optimise la fonction objective.

**Définition 9** (*Problème d’optimisation combinatoire*)

Un problème d’optimisation combinatoire consiste un ensemble d'alternatives qui satisfont une certaine propriété, choisir celle qui optimise une certaine fonction de coût.

Définit à partir d’un triplet (S, p, f) tel que :

* p est un prédicat sur E, i.e. une fonction de E dans {vrai, f aux} ;
* S : l'ensemble des alternatives de solutions ;
* f : S → IR une fonction de coût ou fonction objectif à minimiser (si gain à maximiser, considérer g = −f) ;
* Trouver s\*∈ S tel que f (x∗) = minx ∈ S f (s) (notation s∗ = arg min (f)), f(s) < f(s\*).

p permet de créer un ensemble Sa = {x ∈ S tel que P(x) est vrai}. L’ensemble Sa est appelé l’ensemble des solutions admissibles du problème.

Les problèmes d’optimisation combinatoire sont en général très coûteux à résoudre de façon optimale. C’est en particulier le cas du partitionnement de graphe, dont nous verrons, grâce à la théorie de la complexité, qu’il est NP-difficile.

Lorsque le problème n’est pas soumis à aucune contraintes, le problème d’optimisation combinatoire, vise à trouver une partition des sommets d’un graphe G = (S, A) en k parties de tailles égales (on choisit k diviseur de card(S)), aura pour ensemble de solutions E l’ensemble des partitions de S dont le nombre de parties va d’un au nombre d’éléments de S, et dont les parties sont de tailles quelconques. Par contre, l’ensemble des solutions admissibles du problème, Ea, doit tenir compte des contraintes de celui-ci. Ainsi, dans notre exemple, l’ensemble Ea sera constitué des partitions de S en k parties de tailles égales.

**Définition 10** (*Optimum global, optimum local*)

Soit un problème d’optimisation combinatoire (S, p, f) et Sa l’ensemble des solutions admissibles du problème induit par p. Soit ӯ ∈ Sa.

* si l’on peut prouver que ∀y ∈ Sa, f(ӯ) ≤ f(y), alors on dira que ӯ est l’optimum (minimum) global du problème ;
* s’il existe un ensemble V ⊂ Sa, contenant ӯ, et au moins deux éléments, tel que ∀y ∈ V, f(ӯ) ≤ f(y), alors on dira que ӯ est un optimum (minimum) local du problème

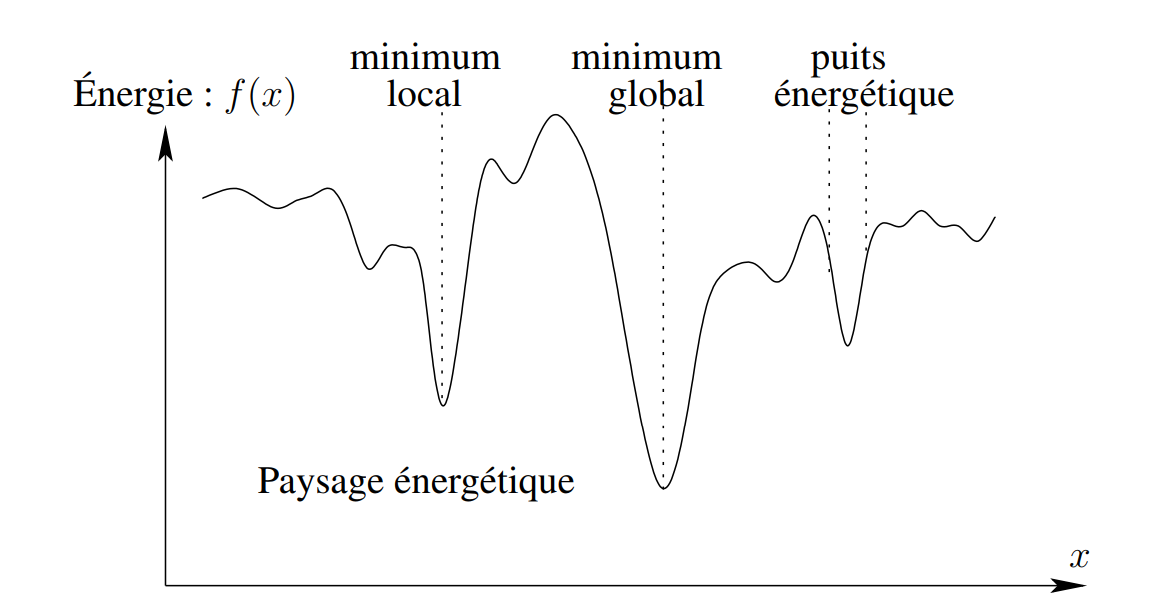


Figure 1 Paysage énergétique dans le cadre continu d’une fonction de coût pour un espace des solutions [BICHOT, 2007]

## ****Théorie de Graphe****

La théorie des graphes est la discipline mathématique et Informatique qui étudie les graphes ayant des problématiques et des démarches scientifiques propres *[François Queyroi, 2013]***.**

Nous présentons quelque lexique employé dans ce domaine.

**Définition 1 (*Graphe*)**

Un graphe est couple **G= (S, A)**, formé d’un ensemble **S** de sommets ou nœuds ou points et d'un ensemble **A** d’arêtes, d’arcs ou lignes qui sont associés à des sous-ensembles à deux éléments de **S**. Les sommets appartenant à une arête sont ses extrémités. Un sommet d'un graphe n'est pas nécessairement extrémité d'une arête : c'est alors un sommet isolé. La taille d'un graphe est, selon le cas, le nombre de ses sommets **|S|** ou le nombre de ses arêtes **|A|**. Selon la nature de l’ensemble A, le graphe peut être orienté ou non orienté.

**Définition 2 (*Graphe orienté, non orienté*)**

Soit un graphe G = (S, A). Si ∀(x, y) ∈ A, (y, x) ∈ A, alors le graphe est dit non orienté et les éléments de A sont appelés arêtes du graphe. Dans ce cas, on note indifféremment une arête : a ∈ A, (s, s’) ∈ A avec s et s’ dans S, ou encore (s’, s). Dans le cas contraire, le graphe est dit orienté et les éléments de A sont appelés arcs du graphe.

**Définition 3 (*Adjacence*)**

Soient un graphe G = (S, A) et une arête a = (s, s’) ∈ A. On dit que les sommets s et s’ sont les sommets adjacents (voisins) à l’arête a. De même, on dit qu’a est l’arête adjacente aux sommets s et s’.

**Définition 4** (*Graphe Complet*)

Un graphe est complet si tous ses sommets sont adjacents entre eux. Deux sommets non adjacents peuvent malgré tout avoir une certaine connections. Dans ce cas, nous parlerons de chemin.

**Définition 5 (***Chemin***)**

Un chemin dans G = (S, A) de longueur n est une suite de sommets a1a2 . . . anan+1 reliés entre eux par des arêtes, c’est-à-dire aiai+1 ∈ A, pour i = 1, 2, . . ., n. Un cycle est un chemin fermé,

C’est-à-dire que a1 = an+1.

**Définition 6 (*Graphe connexe*)**

**Soit un graphe G = (S, A). On dit que ce graphe est connexe si, quels que soient les sommets s et s’ de S, il existe un chemin de s vers s’.**

**Définition 7 (*Boucle et arête multiple*)**

**Une arête est appelée une boucle si ses deux extrémités sont identiques. Si deux arêtes possèdent les mêmes extrémités, alors on dit que l’arête est multiple et que ces deux arêtes sont parallèles. Dans ce cas, la multiplicité d’une arête est le nombre total de ses arêtes parallèles, y compris elle-même.**

**Définition 8 (*Graphe value ou pondéré*)**

**Un graphe pondère est un graphe étiqueté où chaque arêtes (arcs) est affectée d'un nombre réel positif, appelé poids de cette arête.**

**Définition 9 (*Appariement*)**

**Un appariement d’un graphe est un ensemble d’arêtes de ce graphe qui n’ont pas de sommets en commun autrement dit ensemble d’arêtes non-adjacentes deux à deux.**

**Définition 10(***Sous-graphe*)

Un sous-graphe H = (SH, AH) d’un graphe G = (S, A) est le graphe G auquel des sommets et/ou des arêtes ont été enlevés, c’est-à-dire SH ⊆ S et AH ⊆ A.

**Définition 11 (***Chaîne*)

Une chaîne est une suite d’arcs partant d’un sommet x et se terminant à un sommet y. Nous pouvons la définir comme suit : G = (S, A) | X = {1, 2, ..., n} ∧ {(1, 2), (2, 3), ..., (n-1, n)} où x = (1, 2) et y = (n-1, n)

**Définition 12 (***Cycle*)

Un cycle ou circuit est une suite d’arcs partant et finissant au même sommet. Nous pouvons le définir comme suit : G = (S, A) | X = {1, 2, ..., n} ∧ {(1, 2), (2, 3), ..., (n-1, n), (n, 1)} Notons également qu’un cycle de type déterminé est automatiquement une chaîne du même type.

## ****Théorie de jeux****

La théorie des jeux appelée aussi théorie de décision en interaction, se propose d'étudier des situations (appelées jeux) où des individus (les joueurs) prennent des décisions, chacun étant conscient que le résultat de son propre choix (ses gains) dépend de celui des autres. Ces décisions ayant pour but un gain maximum ou un gain stabilisé.

La théorie des jeux est un domaine qui inspire des mathématiques(probabilités, optimisation/contrôle, combinatoire, logique, calculabilité, complexité, analyse/EDP ou encore (en dehors ou en marge des mathématiques)) et d’autre science(économie, cryptographie, physique quantique, cybernétique, biologie, sociologie, linguistique, philosophie), Les premiers fondements mathématiques de ce domaine étaient décrits autour des années 1920 par [Ernst Zermelo](https://fr.wikipedia.org/wiki/Ernst_Zermelo) *[Ernst Zermelo, 1913]*, [Émile Borel](https://fr.wikipedia.org/wiki/%C3%89mile_Borel) *[Émile Borel, 1921]*et John von Neumann *[John von Neumann, 1928].* Les idées de la théorie des jeux sont ensuite développées par [Oskar Morgenstern](https://fr.wikipedia.org/wiki/Oskar_Morgenstern) et John von Neumann en 1944 dans leur ouvrage Theory of Games and Economic Behavior*[J&O, 1944]*.

Dans ce qui suit nous présentons les concepts et vocabulaires qui lui sont propres.

**Définition 1 (**Stratégie)

Une stratégie d’un joueur est la fonction par laquelle il choisit son coup à jouer en fonction de l’état du jeu (ou de la fonction de l’état qui lui est présentée), et d’aléa éventuel. On peut ainsi résumer le jeu en : chaque joueur choisit une stratégie, et la règle du jeu définit alors un gain pour chaque joueur. Les stratégies peuvent être contraintes de différentes manières (par exemple : être calculables par une machine de Turing). Une stratégie est dite gagnante si le joueur qui l’utilise gagne le jeu (supposé avoir une notion de « joueur gagnant ») quels que soient les coups choisis par l’autre joueur.

**Définition 1 (**Stratégie pure)

Une stratégie pure du joueur i est un plan d’action qui prescrit une action de ce joueur pour chaque fois qu’il est susceptible de jouer. On note par Si l’ensemble des stratégies pures du joueur i et par si une stratégie pure de ce joueur.

**Définition 2 (**Stratégie mixte)

Une stratégie mixte du joueur i est une distribution de probabilités pi définie sur l’ensemble des stratégies pures du joueur i. On note Σi l’ensemble des stratégies mixtes du joueur i et par σi une stratégie mixte de ce joueur.

**Définition 3 (**Stratégie locale)

Une stratégie locale du joueur i en un ensemble d’information A est une distribution de probabilités sur l’ensemble des actions disponibles en cet ensemble d’information. On note ΠiA l’ensemble des stratégies locales du joueur i pour l’ensemble d’information A et πiA une stratégie locale de ce joueur en A.

**Définition 4 (**Stratégie comportementale)

Une stratégie comportementale du joueur i est un vecteur de stratégies locales de ce joueur, contenant une stratégie locale par ensemble d’information de ce joueur. On note Πi l’ensemble des stratégies comportementales du joueur i, et πi une stratégie comportementale de ce joueur.

**Définition 1 (**Jeux sous forme stratégique)

Un jeu sous forme stratégique est défini par :

* Un ensemble N = {1, . . ., n} de joueurs.
* Pour chaque joueur i un ensemble de stratégies Si = {s1, . . ., sni}.
* Pour chaque joueur i une fonction dévaluation µi : S1 × . . . × Sn → IR, qui à chaque ensemble de stratégies associe les gains du joueur i.

**Définition 1 (**profil stratégique)

Un profil stratégique (s1\*, …, sn\*) est une solution d’un jeu si on peut justifier que des joueurs rationnels, guidés par leur intérêt personnel, le jouerait.

**Définition 1 (**Stratégie dominante)

On dit qu’une stratégie si\* d’un joueur est une stratégie dominante si, quel que soit le profil des stratégies (s1, …, si-1, si+1, … sn) des autres joueurs, le gain du joueur est maximum lorsqu’il joue cette stratégie.

**Définition 1 (**équilibre en stratégies)

On dit qu’un jeu possède un équilibre en stratégies dominantes s’il admet un profil stratégique (s1\*, …, sn\*) composé uniquement de stratégies dominantes des joueurs.

**Définition 1 (**équilibre de Nash)

On dit qu’un jeu possède un équilibre de Nash s’il admet un profil stratégique (s1\*, …, sn\*), tel que chaque stratégie individuelle de ce profil est une meilleure réponse aux stratégies des autres joueurs.

**Définition 1 (**)

## ****Partitionnement de graphe****

**Le partitionnement de graphe est un problème NP-Complet, trouver les partitions optimales n’est pas une tâche facile à obtenir dans le temps, pour cela deux famille d’algorithme s’impose : les algorithmes avec contrainte et sans contrainte**

# ****Apprentissage Automatique****

**L'apprentissage automatique (en anglais machine learning, littéralement « l'apprentissage machine ») ou apprentissage statistique est un des champs d'étude de l'**[**intelligence artificielle**](http://dictionnaire.sensagent.leparisien.fr/Intelligence%20artificielle/fr-fr/)**, est la discipline scientifique concernée par le développement, l'analyse et l'implémentation de méthodes automatisables qui permettent à une machine (au sens large) d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens**[**algorithmiques**](http://dictionnaire.sensagent.leparisien.fr/Algorithmique/fr-fr/)**plus classiques. Il permet** optimiser un modèle de traitement de l’information selon certains critères de performance à partir d’observations, que ce soit des données-exemples ou des expériences passées. On exerce le processus d’apprentissage lorsque le bon modèle de traitement à utiliser n’est pas connu d’avance. **L'apprentissage automatique est utile lorsque : le problème dépasse l’expertise de l’être humain ou trop peu performante,** Les solutions au problème changent dans le temps, Les solutions doivent être personnalisées etc.

## ****Histoire****

Les robots et automates sont une source d’intérêt depuis plusieurs siècles. Déjà les écrivains de la période romantique traitaient de l’intelligence artificielle et aujourd’hui encore, nous restons fascinés par les robots, que ce soit dans les films, les livres ou les jeux vidéo. La relation de l’être humain à la machine pensante a toujours oscillé **entre crainte et fascination**. Cependant, les réels progrès du machine learning ne commencèrent pas avant les années 50, à une époque où les ordinateurs n’en étaient encore qu’à leurs balbutiements et ou l’intelligence artificielle ne pouvait que faire rêver. Au cours des deux siècles précédents, des théoriciens comme Thomas Bayes, Adrien Marie Legendre et Pierre-Simon Laplace avaient déjà jeté les bases de la recherche, mais il faut attendre les travaux d’**Alan Turing**pour parler concrètement d’apprentissage automatique des machines.

En 1950, Turing a développé une proposition de test d’intelligence artificielle : le **test de Turing**. Il s’agit d’une sorte de jeu dans lequel un ordinateur prétend être un être humain, il imite la conversation humaine. Si la personne n’est pas capable d‘identifier lequel de ses interlocuteurs est une machine, on peut alors considérer que l’ordinateur a passé le test avec succès. Deux ans plus tard, Arthur Samuel a développé un ordinateur qui pouvait jouer aux dames tout en s’améliorant à chaque partie. Le programme avait donc la capacité d’apprendre. Enfin, en 1957, Frank Rosenblatt développa le Perzeptron, un premier algorithme d’apprentissage, il s’agit d’un **réseau neuronal artificiel**.

Dès lors, les scientifiques ont commencé à confier à leurs ordinateurs des épreuves de plus en plus complexes, les machines les maitrisant plus ou moins bien. IBM a développé **Watson**, un programme informatique qui possède un immense référentiel de connaissances et qui peut répondre aux questions posées en langage naturel. Ainsi, il a même participé à la célèbre émission de télévision « Jeopardy! », ce qui a eu un impact fort sur les médias car Watson a gagné la manche. (Cet évènement rappel beaucoup la compétition d’échec de 1997 entre le champion du monde Garri Kasparov et un autre ordinateur d’IBM : le Deep Blue. Là aussi la machine est sortie victorieuse.)

Google et Facebook utilisent le machine learning pour mieux comprendre les utilisateurs et offrir davantage de fonctionnalités. **DeepFace**de Facebook peut même désormais identifier les visages sur les images avec un taux de réussite de 97 pourcent. De plus le moteur de recherche géant a déjà considérablement amélioré la reconnaissance vocale du système d’exploitation Android, la recherche de photos sur Google+ et les recommandations vidéo sur YouTube via son **projet GoogleBrain**.

## ****Principe****

Les algorithmes utilisés permettent, dans une certaine mesure, à un système piloté par ordinateur (un robot éventuellement), ou assisté par ordinateur, d'adapter ses analyses et comportements en réponse, en se fondant sur l'analyse de données empiriques provenant d'une base de données ou de capteurs. La difficulté réside dans le fait que l'ensemble de tous les comportements possibles compte tenu de toutes les entrées possibles devient rapidement trop complexe à décrire (on parle d'[explosion combinatoire](http://dictionnaire.sensagent.leparisien.fr/Explosion%20combinatoire/fr-fr/)) dans les langages de programmation disponibles. On confie donc à des programmes le soin d'ajuster un modèle permettant de simplifier cette complexité et de l'utiliser de manière opérationnelle. De plus, ce modèle est adaptatif, de façon à prendre en compte l'évolution de la base des informations pour lesquelles les comportements en réponse ont été validés, ce que l'on appelle apprendre ; ceci permet d'auto-améliorer le système d'analyse ou de réponse (commande adaptative…), ce qui est une des formes que peut prendre l'intelligence artificielle. Ces programmes, selon leur degré de perfectionnement, intègrent éventuellement des capacités de traitement probabiliste des données, d'analyse de données issues de capteurs, de reconnaissance (reconnaissance vocale, reconnaissance de forme, d'écriture, etc.), de [data-mining](http://dictionnaire.sensagent.leparisien.fr/Data-mining/fr-fr/" \o "Data-mining), d'informatique théorique, etc.

## ****Type d’apprentissage****

Les algorithmes d'apprentissage sont classés en fonction du mode d'apprentissage utilisé :

* **L'**[**apprentissage supervisé**](http://dictionnaire.sensagent.leparisien.fr/Apprentissage%20supervis%C3%A9/fr-fr/)

 Si les classes sont prédéterminées et les exemples connus, le système apprend à classer selon un modèle de classement ; on parle alors d'apprentissage supervisé (ou d'[analyse discriminante](http://dictionnaire.sensagent.leparisien.fr/Analyse%20discriminante/fr-fr/)). Un [expert](http://dictionnaire.sensagent.leparisien.fr/Expert/fr-fr/) (ou oracle) doit préalablement étiqueter des exemples. Le processus se passe en deux phases. Lors de la première phase (hors ligne, dite d'apprentissage), il s'agit de déterminer un modèle des données étiquetées. La seconde phase (en ligne, dite de test) consiste à prédire l'étiquette d'une nouvelle donnée, connaissant le modèle préalablement appris. Parfois il est préférable d'associer une donnée non pas à une classe unique, mais une probabilité d'appartenance à chacune des classes prédéterminées (on parle alors d'apprentissage supervisé probabiliste).

**Exemples 1 :**

L'[analyse discriminante linéaire](http://dictionnaire.sensagent.leparisien.fr/Analyse%20discriminante%20lin%C3%A9aire/fr-fr/) ou les [SVM](http://dictionnaire.sensagent.leparisien.fr/Support%20vector%20machine/fr-fr/) en sont des exemples typiques. Autre exemple : en fonction de points communs détectés avec les [symptômes](http://dictionnaire.sensagent.leparisien.fr/Sympt%C3%B4me/fr-fr/) d'autres patients connus (les « exemples »), le système peut catégoriser de nouveaux patients au vu de leurs [analyses médicales](http://dictionnaire.sensagent.leparisien.fr/Analyses%20m%C3%A9dicales/fr-fr/) en [risque](http://dictionnaire.sensagent.leparisien.fr/Risque/fr-fr/) estimé ([probabilité](http://dictionnaire.sensagent.leparisien.fr/Probabilit%C3%A9/fr-fr/)) de développer telle ou telle maladie.

* **L'**[**apprentissage non-supervisé**](http://dictionnaire.sensagent.leparisien.fr/Apprentissage%20non-supervis%C3%A9/fr-fr/) (ou [classification automatique](http://dictionnaire.sensagent.leparisien.fr/Classification%20automatique/fr-fr/)).

 Quand le système ou l'opérateur ne disposent que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé ou [clustering](http://dictionnaire.sensagent.leparisien.fr/Partitionnement%20de%20donn%C3%A9es/fr-fr/" \o "Partitionnement de données). Aucun expert n'est requis. L'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données. Le [partitionnement de données](http://dictionnaire.sensagent.leparisien.fr/Partitionnement%20de%20donn%C3%A9es/fr-fr/), data clustering en anglais, est un algorithme d'apprentissage non supervisé.  
Le système doit ici dans l'espace de description (la somme des données) cibler les données selon leurs attributs disponibles, pour les classer en groupe homogènes d'exemples. La [similarité](http://dictionnaire.sensagent.leparisien.fr/Similarit%C3%A9/fr-fr/) est généralement calculée selon une fonction de distance entre paires d'exemples. C'est ensuite à l'opérateur d'associer ou déduire du [sens](http://dictionnaire.sensagent.leparisien.fr/Sens/fr-fr/) pour chaque groupe et pour les motifs (patterns en anglais) d'apparition de groupes, ou de groupes de groupes, dans leur « espace ». Divers outils mathématiques et [logiciels](http://dictionnaire.sensagent.leparisien.fr/Logiciel/fr-fr/) peuvent l'aider. On parle aussi d'analyse des données en régression (ajustement d'un modèle par une procédure de type moindres carrés ou autre optimisation d'une fonction de coût). Si l'approche est [probabiliste](http://dictionnaire.sensagent.leparisien.fr/Probabiliste/fr-fr/) (c'est-à-dire que chaque exemple, au lieu d'être classé dans une seule classe, est caractérisé par un jeu de probabilités d'appartenance à chacune des classes), on parle alors de « soft clustering » (par opposition au « hard clustering »).  
Cette méthode est souvent source de [sérendipité](http://dictionnaire.sensagent.leparisien.fr/S%C3%A9rendipit%C3%A9/fr-fr/" \o "Sérendipité).

*SVM : Support Vector* Machine

**Exemple 2 :**

Pour un [épidémiologiste](http://dictionnaire.sensagent.leparisien.fr/%C3%89pid%C3%A9miologiste/fr-fr/) qui voudrait dans un ensemble assez large de victimes de [cancers du foie](http://dictionnaire.sensagent.leparisien.fr/Cancer%20du%20foie/fr-fr/) tenter de faire émerger des hypothèses explicatives, l'ordinateur pourrait différencier différents groupes, que l'épidémiologiste chercherait ensuite à associer à divers facteurs explicatifs, origines géographique, [génétique](http://dictionnaire.sensagent.leparisien.fr/G%C3%A9n%C3%A9tique/fr-fr/), habitudes ou pratiques de consommation, expositions à divers agents potentiellement ou effectivement toxiques ([métaux lourds](http://dictionnaire.sensagent.leparisien.fr/M%C3%A9tal%20lourd/fr-fr/), [toxines](http://dictionnaire.sensagent.leparisien.fr/Toxine/fr-fr/) telle que l'[aflatoxine](http://dictionnaire.sensagent.leparisien.fr/Aflatoxine/fr-fr/), etc.).

* **L'**[**apprentissage semi-supervisé**](http://dictionnaire.sensagent.leparisien.fr/Apprentissage%20semi-supervis%C3%A9/fr-fr/)

Effectué de manière probabiliste ou non, il vise à faire apparaître la distribution sous-jacente des « exemples » dans leur espace de description. Il est mis en œuvre quand des données (ou « étiquettes ») manquent… Le modèle doit utiliser des exemples non-étiquetés pouvant néanmoins renseigner.

**Exemple 3 :**

En médecine, il peut constituer une aide au diagnostic ou au choix des moyens les moins onéreux de tests de diagnostic.

* **L'apprentissage partiellement supervisé** (probabiliste ou non)

Quand l'étiquetage des données est partiel. C'est le cas quand un modèle énonce qu'une donnée n'appartient pas à une classe A, mais peut-être à une classe B ou C (A, B et C étant 3 maladies par exemple évoquées dans le cadre d'un [diagnostic différentiel](http://dictionnaire.sensagent.leparisien.fr/Diagnostic%20diff%C3%A9rentiel/fr-fr/)).

* **L'**[**apprentissage par renforcement**](http://dictionnaire.sensagent.leparisien.fr/Apprentissage%20par%20renforcement/fr-fr/)

 L’algorithme apprend un comportement étant donné une observation. L'action de l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme d'apprentissage.

**Exemple 4 :**

L'algorithme de [Q-learning](http://fr.wikipedia.org/w/index.php?title=Q-learning&action=edit&redlink=1" \o "Q-learning (page inexistante))[*[Article2]*](http://dictionnaire.sensagent.leparisien.fr/Apprentissage%20automatique/fr-fr/#cite_note-2) est un exemple classique.

## ****Exemple****

Un système d'apprentissage automatique peut permettre à un [robot](http://dictionnaire.sensagent.leparisien.fr/Robot/fr-fr/) ayant la capacité de bouger ses membres mais ne sachant initialement rien de la coordination des mouvements permettant la marche, d'apprendre à marcher. Le robot commencera par effectuer des mouvements aléatoires, puis, en sélectionnant et privilégiant les mouvements lui permettant d'avancer, mettra peu à peu en place une marche de plus en plus efficace.

La [reconnaissance de caractères](http://dictionnaire.sensagent.leparisien.fr/Reconnaissance%20de%20caract%C3%A8res/fr-fr/) manuscrits est une tâche complexe car deux caractères similaires ne sont jamais exactement égaux. On peut concevoir un système d'apprentissage automatique qui apprend à reconnaître des caractères en observant des « exemples »

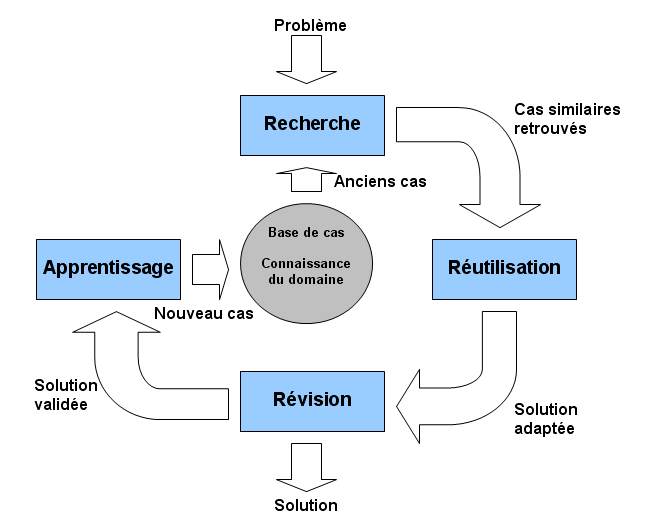
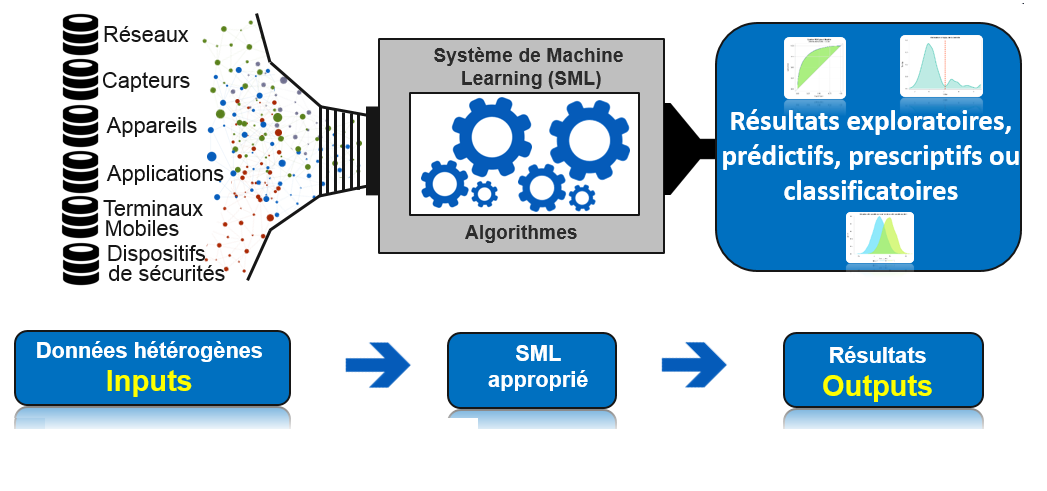
****

Figure 2 Processus d'apprentissage et d'évolution

[Article2] Q-learning, https://en.wikipedia.org/wiki/Q-learning

[Article2] Q-learning, https://en.wikipedi****a.org/wiki/Q-learning

# Base de Donnée Distribuée Orienté Graphe

Le théorème CAP d’Eric A.Brewer affirme qu’une base de données ne peut respecter au plus que 2 propriétés parmi la *cohérence*, la *disponibilité* et la *distribution*.[B01]

Le No SQLétant une solution pour faire face aux 3V (*Volume, Velocity, Variety*), propose de relâcher certaines contraintes lourdes du relationnel (structure des données, langage d'interrogation ou la cohérence) pour favoriser la distribution et obtient les propriétés CP ou AP du théorème CAP. Ainsi selon le type de modèle de données utilisé on distingue 4 modèles de BDD No SQL :

* BDD Dépôt Clés-Valeur
* BDD Clone Big Table
* BDD Orientées Documents
* **BDD Orientées Graphes.**

Une base de donnée orienté graphe permet de modéliser aisément une très grande quantité des données dans des modèles complexes et très connectés tout en offrant une structuration flexible, des opérations de modification et de parcours performante, utilise la théorie de graphe pour stocker, mapper et interroger des relations, intégration d'une [API](https://fr.wikipedia.org/wiki/Interface_de_programmation) utilisant des algorithmes classique de la [Théorie des graphes](https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_graphes). Assure les propriétés ACID des données ce qui permet de répondre avec précision aux interrogations sur les données. Offre une architecture distribuée permettant de stocker un graphe assez large, sans aucun impact sur le temps latence. Ce modèle de donnée apporte une simplicité aux programmateurs de graphes, analystes de donnée, dans la recherche d’une information.

Nous présentons dans la suite le langage de requête de la base de données orientée graphe afin de voir s’il répond aux interrogations CTL.

# Historique

Le premier article sur les graphes est : « Solutio problematis ad geometriam situs pertinensis » écrit par le mathématicien allemand Leonhard Euler (1707-1783), en 1736 en réponse à une curiosité mathématique : comment parcourir une fois et une seule les sept ponts de la ville de Königsberg (Kaliningrad actuellement).

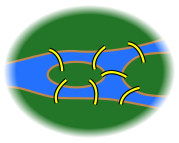
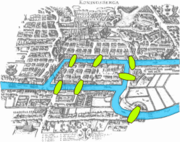
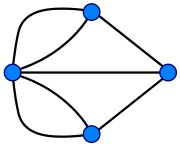
[](https://commons.wikimedia.org/wiki/File:7_bridges.svg?uselang=fr)[](https://commons.wikimedia.org/wiki/File:Konigsberg_bridges.png?uselang=fr)  [](https://commons.wikimedia.org/wiki/File:K%C3%B6nigsberg_graph.svg?uselang=fr)

Figure 3 Modélisation par graphe, des sept ponts de la ville de Königsberg

Graphe, où les points (sommets) représentent les 4 parties de la ville et les liens (arêtes) schématisent les différents ponts. Euler prouve alors qu’il est impossible de traverser chacun des sept ponts de la ville de Königsberg une fois exactement, et de revenir au point de départs, de plus il prouve que pour traverser il aurait fallu pour chaque rive un nombre de pont pair. Ce qui se traduit en théorie des graphes par : pour qu’un trajet passe une fois et une seule sur chaque arête, et revienne au point de départ sur un graphe, il faut que chaque sommet du graphe soit de degré pair.la résolution de ce problème est considéré comme le premier théorème de la théorie des graphes. Au cours du 20ème siècle, la théorie des graphes a connu d'importantes avancées dans diverses disciplines telles que la chimie (isomères), la biologie, les sciences sociales , gestion de projets (C.P.M.), informatique (topologie des réseaux, complexité algorithmique, protocoles de transferts), la physique quantique, l'écologie, etc. Elle constitue une branche à part entière des mathématiques discrètes et d’informatique théorique, grâce aux travaux de König, Menger, Cayley, Erdös, Kuhn, Kruskal, Ford et d'autres. Actuellement, elle a connu un grand regain d'intérêt suite à l'émergence des réseaux sociaux Internet dont les connexions entre "amis" et "suiveurs" constituent des graphes dont l'analyse topologique et statistique peut nous apprendre de nombreuses choses et de plus en plus de domaines lui font appel pour résoudre certaines problèmes réels. A partir d’un graphe Euler répond au problème des sept ponts de Königsberg. À l’inverse des « heureux » mathématiciens qui n’ont besoin que d’une feuille et d’un crayon pour créer, l’ingénieur en informatique utilise des outils tels que des compilateurs et des moteurs de stockage de données pour donner vie à ses idées et de répondre aux problèmes en temps réels.

# Motivation

Aujourd’hui les entreprises croient de manière exponentielle ainsi que la connectivité entre les données, au point que les base de donnée classique ne répondent plus, non seulement à stocker mais aussi ne satisfait pas dans le temps des interrogations voire impossibles dans certains cas de figure : détection d’une anomalie, faire des recommandations, faire collaborer les éléments semblable, cela est devenu complexe par le fait que les données sont difficile à **Sanders**. C’est dans ce cadre de problèmes que les base de donnée orienté graphe apporte leur soutiens pour répondre à l’exigence demander dans le temps, tout en modélisant les données sous forme de graphe ce qui facilite la recherche et le contrôle des donnée.

# Architecture

Les base de donnée orienté graphe offre une quantité de stockage plus de 30 milliard de nœuds et relations dans une machine avec un temps de réponse très faibles lors des interrogations, ainsi pour continuer de traiter autant de donnée possible sans dégradation des performances, recourent à la distribution des graphes, afin de pouvoir traiter un volume de donnée assez complexe dans le temps et s’assurer que le stockage ne sera pas un enfreint. Vu que la distribution est employée, plusieurs machines se mettent en collaboration pour stocker une partie des données tout en référençant la machine dans laquelle résident le nœud externe, ce qui facilite les interrogations sur les données pour un utilisateur car il se rend pas compte que les donne ne réside pas dans la même la machine.

Figure 4 Graphes reparties dans différentes machines

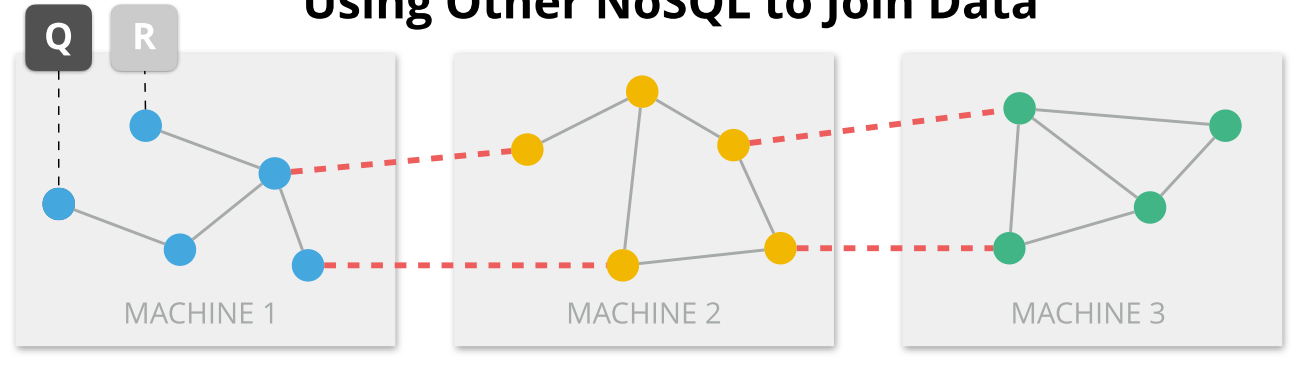
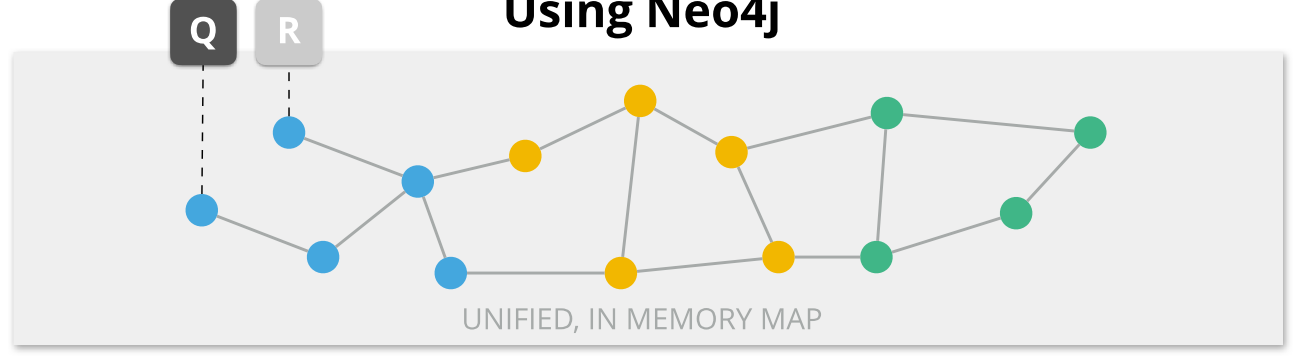


Figure 5 Vue global des graphes distribue



# Logique temporelle arborescente (CTL) & Langage de requete

Les bases de données orienté graphes étant la solution idéal pour repreneur, parcourir un graphe quel que soit la complexité qu’il représente, mais cela n’est pas suffisant d’affirmer qu’il répond à des propriétés CTL pour répondre a cette affirmation il suffit de montrer que le langage cypher permet d’indexer les propriétés CTL dans ce qui suit nous allons considérer la distribution des graphes dans un block.

## Logique temporelle arborescente (CTL)

Logique temporelle arborescente permet de considérer plusieurs futurs possibles à partir d’un état du système plutôt que d’avoir une vue linéaire du système considéré, cela se traduit dans un graphe à chercher les successeurs des états et tirer une conclusion à partir du parcours des états et la propriété à vérifier.

* 1. **Syntaxe**

Les formules de la logique temporelle arborescente sont définies par la grammaire suivante [1, 3] :

1. Les propositions atomiques : φ, Ψ : := p | q | . . . | ture | false φ, Ψ ∈ CTL
2. Les connecteurs booléens : Soit φ, Ψ ∈ LTL, toutes formules de la forme

¬ φ | φ ∧ Ψ | φ ∨ Ψ | φ ⇒ Ψ | φ ⇔ Ψ appartiennent à CTL.

1. Les connecteurs temporels :

Soit φ, Ψ ∈ LTL, toutes formules de la forme EF φ | EG φ | E φU Ψ | EX φ | AF φ | AG φ | A φU Ψ | AX φ appartiennent à CTL.

E, A : quantifications sur toutes les exécutions possibles à partir de l’état courant.

* 1. **Sémantique**

Notons que dans les dessins [2,3] qui suivent l’état vérifier contient la formule.

Ainsi, on définit les opérateurs temporels comme suit :

* **X** (next) **X φ** : "φ est vérifié à l’état suivant"
* **F** (eventually) **F φ** : "il existe un état pour lequel φ est vrai"
* **G** (always) **G φ** : "φ est toujours vérifié dans le futur"
* **E, A** : quantiﬁcations sur toutes les exécutions possibles à partir de l’état courant

|  |  |  |
| --- | --- | --- |
| C:\Users\Etudiant RSD\Pictures\ef.PNG  **EF φ** : « il est possible d’atteindre un état où φ est vérifié » ou « il existe une exécution (opérateur E) conduisant à un état où φ est vérifié (opérateur F)» | **C:\Users\Etudiant RSD\Pictures\af.PNGAF φ** : «φ est vérifié dans le futur» ou « pour toute exécution (opérateur A), il existe un état où φ est vérifié (opérateur F)» | C:\Users\Etudiant RSD\Pictures\ag.PNG **AG φ** : «φ est vérifié pour tout état atteignable» ou « pour toute exécution (opérateur A), φ est toujours vérifié (opérateur G)» |
| **EG φ** : «il existe une exécution (opérateur E) où φ est toujours vérifié (opérateur G)» C:\Users\Etudiant RSD\Pictures\eg.PNG | **C:\Users\Etudiant RSD\Pictures\eU.PNGE φ U Ψ** : «il existe une exécution (opérateur E) durant laquelle φ est vérifié jusqu’à ce que Ψ le soit (φ U Ψ)» | **A φ U Ψ** : «pour toute exécution (opérateur A) φ est vérifié jusqu’à ce que Ψ le soit (φ U Ψ)» C:\Users\Etudiant RSD\Pictures\aU.PNG |
| **EX φ** : «il existe une exécution (opérateur E) dont le prochain état satisfait φ» C:\Users\Etudiant RSD\Pictures\ex.PNG | **AX φ** : «tous les états immédiatement successeur satisfont φ» C:\Users\Etudiant RSD\Pictures\ax.PNG |  |

Tableau 1 Propriétés CTL

* 1. **Exemple**

Pour illustrer l’utilisation de la logique temporelle arborescence dans la formalisation des problèmes pratiques, nous donnons l’exemple du publiphone qui consiste à modéliser les actions qu’un utilisateur exécute pendant l’opération de tentative de téléphoner jusqu’à la fin de la communication. La procédure téléphonée est la suivante :

* L’utilisateur doit décrocher le téléphone.
* Le système affiche insérer une carte.
* L’utilisateur insère une carte.
* Le système affiche le nombre d’unités restantes.
* L’utilisateur compose son numéro de téléphone avec la possibilité de le corriger pour obtenir le bon numéro dans un délai de 2 secondes.
* L’utilisateur communique avec son correspondant, tant qu’il lui reste des unités. – Si la carte est épuisée, l’utilisateur a 10 secondes pour la changer.
* Lorsque la communication est terminée, l’utilisateur raccroche le téléphone.
* Le système affiche retirer la carte. – L’utilisateur retire sa carte.

Cela se traduit en CTL :

u\_decrocher ∧ u\_inserer\_carte ∧

**AX** [(u\_composer\_no ∧ **EF** u\_reprise\_sur\_erreur) ∧

**AX** ((u\_communiquer ∧ **EF** changer\_carte\_vide) **U**

(u\_raccrocher ∧ u\_retirer\_carte))] ⇒ ctl\_telephoner

Pour téléphoner, l’utilisateur doit décrocher le téléphone et insérer une carte (l’ordre de ces deux actions n’intervient pas). Puis, dans tous les états possibles (**opérateur AX**), il composera les numéros, et pour terminer il raccrochera le téléphone et retirera sa carte.

Pour composer les numéros, l’utilisateur pourra (**opérateur E**) se ramener à le corriger (**opérateur F**). Dans tous les états possibles, l’utilisateur communique au téléphone jusqu’à (**opérateur U**) ce qu’il raccroche et retire sa carte avec une possibilité de l’échanger (**opérateur EF).** EF se traduit dans notre cas par éventuellement au lieu de finalement pour “eventually”.

## Langage de Requête

Les base de donnée orienté graphe doivent leur force aux langages de requête permettant d’exprimer dans leur sémantique en quelque prédicats de résoudre des problèmes de décision, de fraude, de recommandation, de classification, graphes. Elle ne permet pas seulement d’interroger mais aussi de mettre à jour des données à la fois compréhensive et efficace. Nous utilisons la sémantique de Neo4j pour construire des requêtes. Ce dernier est construit autour d’un ensemble de clause, operateur, expression, variable et paramètre.

* 1. **Syntaxe**

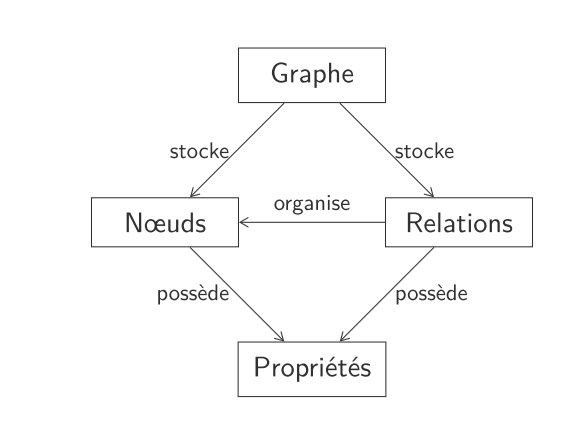
La syntaxe du langage s’inspire des propriétés d’un graphe pour indexée les nœuds. Ainsi un graphe est illustré comme suit :

Figure 6 La structure d'un graphe

1. **Operateurs**

|  |
| --- |
| **operateur General :** DISTINCT, . , [], IN  **Mathématique :** +, -, \*, /, %, ^,abs, ceil, log, sqrt…  **Comparaison :** =, <>, <, >, <=, >=, IS NULL, IS NOT NULL  **Chaine de cratère spécifique de comparaison :** STARTS WITH, ENDS WITH, CONTAINS  **Booléen :** AND, OR, XOR, NOT  **Chaine de caractère** : + (contamination)  **Type de données :** Boolean, Integer, Float, String, List, Map, Node, Relationship, Path |

1. **Expressions**

|  |  |
| --- | --- |
|  | **()**  **(a)** |
| **r** | **() -- () :**  **( ) -[:r]- ( )**  **(a) -- (b)**  **(a) -[:r]- (b)**  **(a) -[?]- (b)**  **() -- > ()**  **() -[:r]- > ()**  **(a) -- > (b)**  **(a) -[:r]- > (b)**  **(a) -[?]- > (b)** |
| … | **(a) -[\*]- > (b)**  **(a) -[\*]- (b)** |
|  | **(a)** **-->** **(b)** **-->**(c)  **(a)** **--> (b) -->** **(c)** **<-- (a)**  **(b) --> () <-- (a)**  **(a) -(\*5)-> (b)**  **(a) -(\*3..5)-> (b)**  **(a)** -**(\*3..)-> (b)**  **(a) -(\*..5)-> (b)**  **(a) -(\*)-> (b)** |

Tableau 2 Expresions reguliers neo4j

(a) - [\*2] **->** (b) équivalent à (a) **-->** () **-->** (b)

(a) - [\*3..5] **->** (b) : longueur entre 3 et 5 (relations)

(a) - [\*3..] **->** (b) : chemin de longueur minimum 3

(a) - [\*..5] **->** (b) : chemin de longueur maximum 5

(a) - [\*] **->** (b) : n'importe quelle longueur

1. **Clauses**

Une clause est une sorte d’opération (primitive) permettant de : créer des nœuds, des relations, chercher des données dans le graphe, filtrer, projeter, trie des données, etc. ainsi selon le type d’opération on distingue 3 catégories de clauses :

* Les clauses de lecture : elles servent à réaliser des opérations de recherche, de filtrage de données dans la base selon un critère spécifique.

On peut citer :

|  |
| --- |
| **Match :** spécifier le pattern à chercher |
| **Where :** permet de filtrer les résultats d'une requête de Match ou Optional Match |
| **Start :** permet de trouver des points de départ (nœud ou Relations). Start concerne seulement les éléments indexés |
| **Load CSV :** permet d’importer un fichier csv |
| **Aggregation:** permet d’utiliser des fonctions d’agrégation **(Count, Sum, AVG, max/min,..).** |

* Les clauses d’écritures : ils permettent de réaliser des opérations de mise à jour des données (nœuds, relations, propriétés).

On peut citer :

|  |
| --- |
| **Create :** créer un nœud ou une relation |
| **Merge :** s’assure que le motif existe, sinon, il faut le créer |
| **Set :** actualiser les labels ou les propriétés des nœuds ou des relations |
| **Delete :** supprimer un nœud, une relation ou un path |
| **Remove :** supprimer des propriétés et des labels d’un nœud ou une relation |
| **Create Unique :** il fait appel à la fois a la clause Match et Create. Il peut créer un nœud s’il n’existe pas. |
| **Load CSV :** importer un fichier csv et ou créer les éléments associés |
| **Foreach :** actualiser les valeurs d’une liste appartenant à un Path ou un résultat d’agrégation |
| **Periodic Commit :** limiter le nombre de lignes par transaction. il est utile pour éviter la surcharge de la mémoire lors d’un chargement de fichier csv. |

* Les clauses générales : ce sont des opérations qui ne font partie ni de clauses de lecture ni de clause d’écriture, mais les utilisent.

|  |
| --- |
| **Return :** permet de définir les données à inclure dans le résultat de la requête. |
| **With :** permet d’enchainer des requêtes, similaire à pipe. Par conséquent le résultat d’une requête pourrait être accessible pour la seconde requête. |
| **Order by :** permet de trier le résultat de retour, similaire à order by de SQL. Il est utilisé avec les clauses Return ou With. |
| **Limit :** permet de limiter le nombre d’éléments retournés par la clause Return. Elle est similaire à la clause Limit de SQL. |
| **Skip :** comme en SQL, il permet de retourner les résultats à partir d’une ligne donnée. |
| **Unwind :** permet de transformer une liste d’éléments en une séquence de lignes. La liste peut être des paramètres ou des résultats. |
| **Union :** comme en SQL, cette clause permet de fusionner des jeux de données sans répétition. |
| **Call :** permet d’invoquer une procédure déployée sur le serveur Neo4j. |

* 1. **Sémantique**

La sémantique des requetés dépend du besoin, et de l’habilité du programmeur, de ce fait nous proposons des séries d’exemples similaire avec aux CTL afin de pouvoir réaliser des requêtes parfaitement équivalent au CTL. Des graphes serons employé pour montrer le rendu de la requête.

START <lookup> MATCH <pattern> RETURN <expr>

MATCH <pattern> WHERE <condition> RETURN <expr>

|  |  |  |  |
| --- | --- | --- | --- |
| H:\GlobalSchema.png | | | |
| MATCH p=(a:Company)-[:Subcontractor\*]->(a)  RETURN a, p | | (C1)-->(C2)-->(C3)-->(C1) | |
| MATCH (A1:Company)-[:Subcontractor\*]->(A2:Company),  (A1)-->(:Denomination\_group)<--(A2),  (A1)-->(:Street)<--(A2)  RETURN … | | (C10)-->(C3)-->(C1)-->(C2)-->(C9) | |
| E:\Master1 RSD\Memoire\img\Capture17.PNG | | |
| MATCH p = (charlie:Person)-[\* {blocked:false}]-(martin:Person) WHERE charlie.name = 'Charlie Sheen' AND martin.name = 'Martin Sheen' RETURN p | (name:"Charlie Sheen")-[blocked:false]- ( ) –[blocked:false] – (name:"Martin Sheen") | |
| MATCH p = (michael {name: 'Michael Douglas'})-[]->() RETURN p | (name:"Michael Douglas") – [role:"President Andrew Shepherd"] - >  (title:"The American President")  (name:"Michael Douglas") –[role:"Gordon Gekko"] - > (title:"Wall Street") | |
| MATCH (martin:Person {name: 'Martin Sheen'}), (oliver:Person {name: 'Oliver Stone'}),  p = shortestPath((martin)-[\*..15]-(oliver))  RETURN p | (name:"Martin Sheen") –[role:"Carl Fox"] – (title:"Wall Street") –[]- (name:"Oliver Stone") | |
| C:\Users\Etudiant RSD\Pictures\ex.PNG  START state0 = node(1) MATCH state –[?]->( stat1 {name: 'φ '})  Where exist(state1) LIMIT 1  Return toBoolean('TRUE') | C:\Users\Etudiant RSD\Pictures\ax.PNG  START state0 = node(1) MATCH s=state –[?]->( stat1 {name: 'φ '})  Where exist(state1)  Return s | |
| START state0 = node(1)  C:\Users\Etudiant RSD\Pictures\af.PNGMATCH (s1:state)-[:Subcontractor\*]->(s2:state),  (s1)-->( (stat3 c)<--(s2),  RETURN state3 | C:\Users\Etudiant RSD\Pictures\ag.PNGSTART state0 = node(1): {name: 'φ '} MATCH s=state0  -[:Subcontractor\*] -> ( stat1 :{name: 'φ '})  Where not exist( **(state0)-->( )<--( stat1)** )  Return s | |

Tableau 3 Séries d'exemple de requêtes neo4j

# Conclusion

La modélisation d’un problème en graphe, permet de visualiser les éventuels solutions offert, cela est encore plus fiable avec les bases de données graphes, qui disposent un langage de requête assez puissant pour exprimer des interrogations. Il aussi possible de former des interrogations de CTL.

# Principe Modèle Checker Distribué

Le modèle checker distribué proposé par Bouneb Zine *[Bouneb, 2011]* permet des vérifications parallèles d’une formule φ sur les fragments de la structure de kripke distribuée. Du fait de la distribution, la vérification de φ de certains états *S* dépend de la vérité en *S’* de cette formule, *S’ hébergé* dans une autre machine (*mj*).La machine *mi* effectue la vérification de φ sur le fragment détenue en considérant la valeur logique de *S’* indécidable L (*S’*, φ)=┴ car la formule φ peut-être vérifier en *S’.* Lorsque la machine *mj* termine le calcul, si la formule est vérifié en *S’aucune* action n’est envoyé à *mi*, alors machine *mi* considère que la formule est vérifiée en *S’*, sinon *mj* précise à *mi* la valeur logique de S’ qui est L (*S’*, φ) =false alors *mi* reprend le calcul avec la valeur précisé pour déterminer la vérité des autres états. Une fois que la terminaison est détectée alors la machine détenteur de l’état initial déduit la véracité de φ.

A ce principe on ajoute des actions à exécuter pendant le calcul de vérification pour étiqueter un état. Les étiquètes ajouter à l’état indique : la cause lorsqu’une formule n’est pas vérifier en *S*, cela peut-être en *S* ou *S’;* et aussi le nombre de calcule qui a permis de détecter la vérité de *S.* A partir de ces informations de plus il est susceptible de réaliser des études de réorganisation des espace d’états pour accélérer le temps de calcule.

**Exemple d’application**

Soit une structure de kripke distribué sur 3 machines, pour ces espaces distribués on applique les principes précèdent pour vérifier la formule suivant : AG (aVc).

Figure Structure de kripke distribué

┴

┴

b

a

S1

S2

S3

S4

**Machine : M1**

S4

┴

b,c

S2

**Machine : M2**

S5

S3

S4

┴

┴

c

a,b,c

S1

**Machine : M3**

Résultats de vérification des formules sont représentés dans le tableau ci-dessous dont les lignes représentent les données concernant un état et les colonnes les types de donnés construit durant le processus de vérification. Les abréviations utilisées sont :

* **Id** : identifiant de l’état,
* **T** : le type d’état (I : interne ; E : externe),
* **ME** : Machine à solliciter pour un état Externe,
* **RF**: Cause de la non validité de la formule (c : chemin/ e : état),
* **NR** : nombre de recalcule permettant de déterminer la valeur logique,
* **VL** : valeur logique de la formule a vérifié (true : 1 ; false : 0 ; **┴** : -1).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Itération 1 | | | | | | | Itération2 | | | | | | |
| Id | T | VL | NR | RF | M | ME | Id | T | VL | NR | RF | M | ME |
| S1 | I | 0 | 1 | E | M1 |  |  |  |  |  |  |  |  |
| S2 | I | -1 | 1 |  | M1 |  |  |  |  |  |  |  |  |
| S3 | E | -1 | 1 |  | M1 | M3 |  |  |  |  |  |  |  |
| S4 | E | -1 | 1 |  | M1 | M2 |  |  |  |  |  |  |  |
| S2 | E | -1 | 1 |  | M2 | M1 |  |  |  |  |  |  |  |
| S4 | I | -1 | 1 |  | M2 |  |  |  |  |  |  |  |  |
| S4 | E | -1 | 1 |  | M3 | M2 |  |  |  |  |  |  |  |
| S5 | I | -1 | 1 |  | M3 |  | S5 | I | 0 | 2 | C | M3 |  |
| S3 | I | -1 | 1 |  | M3 |  |  |  |  |  |  |  |  |
| S1 | E | -1 | 1 | E | M3 | M1 | S1 | E | 0 | 2 | E | M3 | M1 |

Tableau 4 Statistique des états

Après la première itération la machine m1 constate que la formule n’est pas vérifiée sur l’état S1, cela est signaler à la machine m3 qui soliste cette état. Il prend en compte la modification et démarre le processus de vérification sur le fragment, la vérité de l’état S5 bascule d’indésirable à false à la deuxième itération, après cette itération aucune action de modification n’a été signalé alors la machine m3 déduit la vérité de la formule.

Jusqu’à ce point, on fait qu’appliquer le modèle checker pour vérifier une formule, sauf que des informations ont été ajouté à l’espace d’états, à partir de ces informations générer on se fixe des objectives optimisations sur le traitement a effectué.

# Paramètre d’optimisation

Après une analyse du principe précèdent on constate que la distribution des états lié entrainent une augmentation du temps de calcul car vue leur distribution un changement d’états affecte les autres, ce qui entraine un nouveau calcul de vérification dans les machines sollicitant l’état. A ce problème la diminution leur distribution améliore le temps de traitement. Ainsi pour minimiser la distribution des espaces états fortement lié affectant le temps de traitement, des objectifs sont apprendre à considération pour garantir la distribution des espaces états et non leur centralisation car celui-ci ressoude totalement le problème. L’objectif fixé est d’assurer que :

* **α** : le temps de traitement a minimisé si possible ;
* **β**: l’équilibrage de charge entre les machines ;
* **γ**: les dupliquas d’une machines.

La prise en considération simultanément des 3 paramètres fixés permet de minimiser à la fois le temps de traitement et garantir un l’équilibrage de charge. La résolution de cette équation entrain l’utilisation de fonctions objectives pour trouver l’espace de solution possible car il peut exister une infinité de configurations respectant ces contraints. Différente techniques sont offert à l’obtention l’espace de solutions, pour ce contexte la technique d’équilibre de Nash combiné aux fonctions objectives est envisageable.

# Principe d’équilibre de Nash

L'équilibre de Nash est un type de solution proposé par John Forbes Nash en 1950 *[Nash, 1950]* pour la recherche d’une solution optimale. Il est couramment utilisé en théorie des jeux, présente une combinaison de décisions individuelles, appelées « stratégies », où chacun anticipe correctement les choix des autres ; il y a autoréalisation, puisque l'issue réalisée est le fruit de décisions prises en pensant qu'elle va se réaliser. En théorie des jeux la question que se pose un joueur au moment de faire son choix est : que va faire l'autre ? Ses croyances concernant le comportement des autres ont donc un rôle essentiel au moment de la décision. La diversité de croyance correspondre ainsi à une multiplicité d'équilibres. Dans les jeux coopératifs on autorise la communication et les accords entre joueurs avant la partie, les messages formulés par un joueur sont transmis sans modification à l'autre joueur, les accords entre joueurs seront respectés, ces hypothèses permet d’obtenir un équilibre de Nash.

En utilisant le principe de Nash, J.A Désidéri *[Désidéri, 2007]* propose un algorithme de partitionnement de territoire qui se base sur des fonctions objectives. Il considère le cas où on dispose des fonctionnels prépondérants, ainsi l’algorithme cherche à optimiser les fonctions prioritaires avec une moindre dégradation tout en associe aux fonctionnelles secondaire des paramètres qui engendre des grandes variations. La recherche de l’équilibre de Nash se fait par échange des résultats obtenus pour chaque fonction objective travaillant avec une partie seulement des variables, les autres étant fixées par les résultats obtenue pour les autres fonctions objectives. Cet équilibre est atteint quand l’optimisation de chaque fonction conduit toujours à la même solution.

En s’inspirant de ces principes, on propose une stratégie d’optimisation des paramètres défini ci-dessus.

# Principe de Optimisation (α, β, γ)

Après le passage du modèle checker, les états présentent des informations par rapport à la vérification. L’analyse de ces résultats, montre que la distribution des états fortement lié augmente le temps de traitement α (paramètres d’optimisation), dans l’exemple précédant l’état S1 et S5 sont lié leur distribution avait augmenté le temps de traitement, et supposant dans le cas d’un exemple complexe ou un ensemble d’états lié sont distribué dans ces situations le temps de traitement est élevé. L’idée de faire basculer tous ces états dans une machine peut conduire à une dégradation des paramètres (β, γ). L’optimisation des paramètres (α, β, γ) reviens à simuler un jeu coopératif entre les machine pour un but commun.

La coopération des machines démarre avec une prise de contact sur les valeurs initiales à savoir le nombre d’états stocker dans la machine(β), le nombre de dupliquât des états de l’autre machine qui sont présent(γ), à partir de ces valeurs il est possible de calculer le nombre minimum et maximum d’état qui sont susceptible d’être stocker dans chaque machine. Le calcule de cette intervalle entre 2 machine est comme suit :

* On note (β1, γ1) les valeurs de la machine m1 respectivement pour m2 (β2, γ2).
* On note le moyen des états.
* L’écart δ est
* Le nombre minimum d’états est :
* Le nombre maximum d’états est : ;
* Chaque machine doit contenir un nombre d’états β tel que : .

Par ailler on sait que deux machines sont liées par la dépendance d’un ensemble d’états, du fait de cette dépendance certain états peuvent être fortement lié et retarder la vérification lorsque la formule n’est pas valide sur ces états c’était le cas dans l’exemple précédente. Lorsqu’une formule n’est pas valide sur un état deux cas s’impose soit il n’est pas valide sur l’état lui-même ou sur ces successeurs, la minimisation du temps de calcul(α) revient à trouver le nombre des états impliqué. D’après le principe du modèle checker, lorsque la formule est invalide sur un état sollicité alors celui-ci notifie les machines qui la sollicite, le travail de recherche des états impliqué peut être partager en deux : d’une part par la machine sollicitée retrouver les états successeur à partir des quels le problème avait débuté et de l’autre côté les machine retrouve les états prédécesseur qui seront impliqué. Lors de ce parcourt on suppose que ce le nombre d’états qui sont susceptible d’être déplacer vers l’autre machine d’où l’incrémentation la valeur de β (l’autre machine) pour chaque état rencontrer, cette limite est trouvée avec un nombre d’états susceptible à être dupliquer, ce nombre est associé à γ de l’autre machine. A la fin de ce processus on obtient α minimiser avec valeur pour β, γ.

On modélise ce scénario sur l’exemple suivant:

S1

a

a

a

┴

a

a

a

┴

┴

a

S2

S3

S5

S8

S10

S9

S6

S7

S11

b

a

┴

┴

a

b

┴

a

a

S9

S7

S11

S13

S22

S15

S14

S12

S10

a

a

a

a

a

b

a

┴

a

┴

S15

S17

S19

S21

S22

S2

S20

S11

S18

S16

Figure Structure Kripk distribué

* 1. **Algorithmes**

Dans cette section on détails le processus de partitionnement des espaces états entre les machines. Il comporte 2 étapes : la première étape consiste à calculer le nombre d’états à déplacer et le nombre de dupliquât possible pour ce déplacement par rapport à un état qui présente une valeur logique false, la deuxième étape consiste à appliquer une stratégie qui optimise les paramètres fixés.

1. Algorithme model checking

L’algorithme du model checking définie au sens de Kleene[Bouneb,2011] est :

1. Etape 1

Cette étape est générée au moment du processus de model checker pour énumérer les états liés à un autre état sur lequel la propriété n’est pas vérifiée. Sur ces états dépendants la propriété est ainsi vérifiée mais il n’est pas toujours vrai sur les chemins alors la dépendance est soit direct ou indirect. Lorsque la propriété n’est pas vérifiée pas à un état alors on énumère le nombre d’états qui lui sont dépendant, la limite de ces états peut entrainer des duplications de certains états.

Dans ce qui suit on définit la démarche d’énumération des états, en commençant par l’initialisation des paramètres, ensuite l’énumération, ces méthodes sont ajoutées à l’algorithme de model checker définit ci-dessus.

Pendant l’itération encours, les états détecter qui ne contiennent pas la propriété à cet instant, alors les paramètres (β, γ) sont à initialiser à zéro, si l’état appartient à la machine alors γ à 1 cela signifie que la duplication de cet état est suffisant. La formalisation de ce principe est faite au niveau de l’équation (3) on obtient:

Pour alors

Si

finsi

finpour

**Exemple**

L’exécution du formule AG(a) sur la structure de kripke **figure 8**, les résultats d’initialisation sont:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** |  |  | **I** | **M** | **T** |
| S9 | 0 | 1 | 1 | M2 | I |
| S14 | 0 | 1 | 1 | M2 | I |
| S19 | 0 | 1 | 1 | M3 | I |

Tableau 5 Initialisation itération 1

A la première itération seule les états locaux qui ont été initialisé, car la propriété n’est pas vérifiée sur eux.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Itération 2 | | | | | |
| **Id** |  |  | **I** | **M** | **T** |
| S9 | 0 | 0 | 2 | M1 | E |
| S15 | 0 | 0 | 2 | M2 | E |
| S10 | 0 | 0 | 2 | M1 | E |
| Itération 3 | | | | | |
| S7 | 0 | 0 | 3 | M2 | E |
| S2 | 0 | 0 | 3 | M3 | E |
| Itération 4 | | | | | |
| S22 | 0 | 0 | 4 | M2 | E |
| Itération 5 | | | | | |
| S11 | 0 | 0 | 5 | M1 | E |
| S11 | 0 | 0 | 5 | M3 | E |

Tableau 6 Initialisation itération

A partir de l’itération 2, l’initialisation concerne que les états externes, car la redistribution des espaces états fait en sorte que le changement n’est pas instantané alors il faut attendre une notification de changement sur un border pour reprendre le calcule alors certains états notifier retrouvent que la formule n’est pas vérifiée sur leur chemin, selon la propriété ils deviennent fausse et notifie les machines qui sollicite celui-ci.

Apres l’étape d’initialisation, on marque les états qui peuvent être fausse durant l’itération encours par le nouvel état qu’il dépend c’est-à-dire pour tout états trouvé on lui ajoute le nouvel état à son ensemble de dépendance. Pour se faire on modifie les 2 algorithmes de base du model checker, auxquels on ajoute à chacun le principe de marquage, il est défini comme suit :

pour le premier algorithme de base défini à l’équation (15), celui-ci concerne tous les chemins, alors lorsque la formule n’est pas vérifiée à un des successeurs, deux cas se présente : soit la propriété est vérifiée sur l’état alors on lui rajoute les états qu’ils dépendent à son ensemble de dépendance et l’itération encours à l’ensemble des itérations, dans l’autre cas la propriété n’est pas vérifié alors cette état peut être dupliquer soit dans la machine réalisant le traitement ou dans d’autres machines donc la limite de l’effet des autre états sont éteints, ils sont stocké dans la l’ensemble limite de l’état ;

Le deuxième algorithme à l’équation (16) suit le même principe que le précédant sauf que tous les successeurs de l’état doivent être fausse. De plus des condition d’impose chaque algorithme s’ajoute celui de l’itération courant qui doit être inclut à l’état successeur. La formalisation de ce principe est la suivante:

Pour alors

Si or or alors

Sinon

Si alors

Pour alors

e.i;

finpour

Sinon

Pour alors

;

Finpour

Finsi

Finsi

finpour

Pour alors

Si or alors

Sinon

Si alors

Pour alors

e.i;

finpour

Sinon

Pour alors

;

Finpour

Finsi

Finsi

Finpour

**Exemple**

L’exécution du formule AG(a) sur la structure de kripke figure 8,

Les résultats des marquages sont :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **T** | **F** | **I** | **limit** | **fn** | **M** | **Id** | **T** | **f** | **i** | **Limit** | **Fn** | **M** |
| S9 | I |  | 1 | S14 | S14 | M2 | S15 | I | S19 | 1 |  |  | M3 |
| S10 | I | S14 | 1 |  |  | M2 | S16 | I | S19 | 1 |  |  | M3 |
| S12 | I | S14 | 1 |  |  | M2 | S17 | I | S19 | 1 |  |  | M3 |
|  |  |  |  |  |  |  | S18 | I | S19 | 1 |  |  | M3 |

Tableau 7 Marquage itération

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Id** | **T** | **F** | **I** | **Limit** | **fn** | **M** |
| S1 | I | S9,S10 | 2 |  |  | M1 |
| S2 | I | S9,S10 | 2 |  |  | M1 |
| S3 | I | S9,S10 | 2 |  |  | M1 |
| S5 | I | S9,S10 | 2 |  |  | M1 |
| S6 | I | S9 | 2 |  |  | M1 |
| S7 | I | S9 | 2 |  |  | M1 |
| S8 | I | S10 | 2 |  |  | M1 |
| S9 | E |  | 2 |  |  | M1 |
| S10 | E |  | 2 |  |  | M1 |
| S14 | I |  | 1 | S15 | S15 | M2 |
| S15 | E |  | 2 |  |  | M2 |

Tableau 8 Marquage itération 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **T** | **F** | **I** | **Limit** | **fn** | **M** | **Id** | **T** | **F** | **I** | **Limit** | **Fn** | **M** |
| S9 | I |  | 1 | S7 | S7 | M2 | S2 | E |  | 3 |  |  | M3 |
| S7 | E |  | 3 |  |  | M2 | S19 | I | S19 | 1 | S2 | S2 | M3 |
|  |  |  |  |  |  |  | S20 | I | S2 | 3 |  |  | M3 |
|  |  |  |  |  |  |  | S22 | I | S2 | 3 |  |  | M3 |

Tableau 9 Marquage itération 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Id** | **T** | **F** | **I** | **Limit** | **fn** | **M** |
| S11 | I | S22 | 4 |  |  | M2 |
| S13 | I | S22 | 4 |  |  | M2 |
| S14 | I |  |  | S15,S22 | S15,S22 | M2 |
| S22 | E |  | 4 |  |  | M2 |

Tableau 10 Marquage itération 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **T** | **F** | **I** | **Limit** | **Fn** | **M** | **Id** | **T** | **F** | **I** | **Limit** | **Fn** | **M** |
| S1 | I |  | 1 ;5 | S7 | S7 | M2 | S11 | E |  | 1 ;5 |  |  | M3 |
| S2 | I |  | 1 ;5 |  |  | M2 | S15 | I | S19,s11 | 1 ;5 |  |  | M3 |
| S11 | 11 |  | 5 |  |  |  | S16 | I | S19,s11 | 1 ;5 |  |  | M3 |
|  |  |  |  |  |  |  | S18 | I | S19,s11 | 1 ;5 |  |  | M3 |

Tableau 11 Marquage itération 5

**Pour l’itération 1:** on constate que S9 ne dépendant pas de S14 car la propriété n’était pas vérifiée pour cet état. Alors il l’enregistre dans un ensemble limite, par la suite S14 s’en servira pour incrémenter le nombre de dupliqua.

**Pour l’itération 2 :**  on remarque que certains états dépendent de plusieurs états, et aussi s14 n’est pas dépendant de s15 car la propriété n’est pas vérifiée pour cette état, celui-ci a été détecté dans l’itération précédente.

**Pour l’itération 3:** on remarque que s9 qui est indépendant de s7 au niveau de la machine 2 et s19 qui l’est pas aussi avec s2 dans la machine 3.

**Pour l’itération 4:** dans cette itération l’ensemble limite de s14 s’est élargie vue qu’il est indépendant de s22.

**Pour l’itération 5:** on constate que certains états dépendent de plusieurs itérations.

Pour tous les itérations états qui dépendent d’autre états, mentionnent celui-ci dans leurs ensembles de dépendance.

Après avoir marqué les états, intervient la démarche d’énumération, pour attribué des valeurs aux paramètres (β, γ) aux états dépendants alors deux cas se présente à savoir:

les états externe (border) pour lesquelles la formule n’est pas vérifié, la valeur de β correspond au le nombre d’état qui dépendant de lui c’est-à-dire le nombre des états ayant insérer cette état dans leur ensemble de dépendance, ainsi pour cette état la duplication(γm) sera fait dans la machine qui effectue le traitement, cela est dû par la direction que prennent les arcs alors le nombre de β et γm sont incrémenté par le nombre des états qui l’ont inséré dans leur ensemble limite; pour ce qui est des états qui sont sollicités par d’autres machines(Notifer) la valeur de β correspond aux nombres des successeurs direct et indirect dont l’intersection de l’ensemble dépendance de l’état successeur et celui de l’état est diffèrent de vide et le nombre de dupliqua correspond aux nombre des états interne qui sont inclut dans l’ensemble de l’état notifier.

Pour chacun de ces 2 principes, on suppose que la formule n’est pas vérifiée sur ces états, la formalisation de cette démarche est la suivante :

Pour S alors

Si () and () and () alors

Pour alors

Si ) alors

;

Sinon si alors

Finsi

Finpour

Sinon si ( alors

Pour

finpour

Finsi

Finpour

Pour ) alors

Finpour

Finfunction

**Exemple**

L’exécution du formule AG(a) sur la structure de kripke figure 8, les résultats du principe d’énumération sont :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Itération 1** | | | | | | |
| **Id** |  |  |  | **I** | **M** | **T** |
| S9 | 0 | 1 |  | 1 | M2 | Notifier |
| S10 | 2 | 1 |  | 1 | M2 | Notifier |
| S15 | 3 | 1 |  | 1 | M3 | Notifier |
| **Itération 2** | | | | | | |
| S2 | 6 | 0 |  | 2 | M1 | Notifier |
| S7 | 2 | 0 |  | 2 | M1 | Notifier |
| S9 | 6 | 0 |  | 2 | M1 | Border |
| S15 | 1 | 0 | 1 | 2 | M2 | Border |
| S10 | 5 | 0 |  | 2 | M1 | Border |
| **Itération 3** | | | | | | |
| S2 | 3 | 0 | 1 | 3 | M3 | Border |
| S7 | 1 | 0 |  | 3 | M2 | Border |
| S22 | 1 | 0 |  | 3 | M3 | Notifier |
| **Itération 4** | | | | | | |
| S11 | 2 | 0 |  | 4 | M2 | Notifier |
| S2. | 3 |  | 1 | 4 | M2 | Border |
| **Itération 5** | | | | | | |
| S11 | 2 | 0 |  | 5 | M1 | Border |
| S11 | 3 | 0 |  | 5 | M3 | Border |
| S15 | 4 | 1 |  | 5 | M3 | Notifier |

Tableau 12 décompte à l'itération 1

Après une première itération, des valeurs sont associé aux paramètres (β, γ) à certains états (Notifier et Border), cela explique soit ils dépendent d’autres états ou la formule n’est pas vérifié sur l’état. Ces résultats prennent en considérations les valeurs précédentes.

Une deuxième itération prend en compte des résultats obtenue dans les tableaux précédant de la deuxième itération on a le résultat de décompte.

Une fois que le principe de vérification est terminé, alors intervient la coopération des machines pour échanger les valeurs obtenues aux paramètres (β, γ, γm, f). Chaque machine envoi les valeurs calculer pour l’état border à la machine concernée, de même pour les valeurs des états notifier sont envoyés aux machine qui la sollicite. A la réception les valeurs sont sauvegardés au niveau de l’état concerné dans tableau qui représente les valeurs reçues pour une machine. La formalisation est la suivante :

**Envoi de: id, β, γ, γm, f par i**

Pour alors

Si alors

F=(type(e)==Notifier) ? e : Ø;

Pour alors

Finpour

Finsi

finpour

**A la réception( par j**

Pour alors

Si e.id==id alors

;

;

finsi

finpour

Exemple d’envoi et réception des valeurs calculés à l’énumération.

**Pour S9 :**

id :=s9 ;β :=6 ;γ :=0 ;γm=0 ;f :=Ø ;i=1

M1 envoie() à M2 ;

A la réception de() par M2, celui-ci les enregistre sur l’état S9

S9

;

;

;

id :=s9 ;β :=0 ;γ :=1 ;γm=0 ;f :=S9 ;i=2

M2 envoie() à M1 ;

A la réception de() par M1, celui-ci les enregistre sur l’état S9

S9

;

;

;

**Pour S10:**

id :=s10 ;β :=5 ;γ :=0 ;γm=0 ;f :=Ø ;i=1

M1 envoie() à M2 ;

A la réception de() par M2, celui-ci les enregistre sur l’état S10

S10

;

;

;

id :=s10;β :=2 ;γ :=1 ;γm=1 ;f :=S10 ;i=2

M2 envoie() à M1 ;

A la réception de() par M1, celui-ci les enregistre sur l’état S10

S10

;

;

;

1. Etape 2

A la fin de l’étape précédente on obtient des valeurs aux paramètres (β, γ), qu’avait calculé chaque machine sur les états border et notifier. Ces valeurs sont utilisées pour réaliser une stratégie de redistribution des espace d’états. Une stratégie consiste à chercher les états minimums à déplacer et à importer. Ainsi chaque machine recherche les différentes stratégies qui lui conviennent pour entamer le déplacement des espaces états associé aux stratégies.

Dans ce qui suit on définit la démarche de cherche des états minimums à déplacer et à importer, avec l’envoi et réception de ces états.

L’équilibrage décharge est assuré en important un nombre minimum des états, cela revient à rechercher sur les états border et notifier celui qui comporte les valeurs minimales pour les paramètres (β, γ). La recherche est focalisée sur les valeurs calculé par les autres machine et il est inferieur a la valeur local généré , toute foie on met en priorité les états ayant une double dépendance c’est-à-dire le cas où les états notifier dépendent d’autres états externe et ces machines aussi ces états dépendent de cette état notifier ; Le mieux serai de recherche en premier temps les valeurs minimales de ce type d’état s’il existe car cela entraine une diminution de temps calcule de deux itération contrairement à la recherche du minimum les états non cycliques qui diminue le temps de calcule d’une itération. La formalisation de ce principe est la suivante :

Pour alors

Pour alors

Pour alors

Si ) and (e

and (()or()) and

((()or()or ==null)) alors

Finsi

Finpour

Finpour

finpour

Si alors

Pour alors

Pour alors

Si (( )or ( and )) and( ==null or (( )or ( and ))) alors

Finsi

Finpour

Finpour

Finsi

De même, pour les états à déplacer l’équilibrage décharge est assuré en déplaçant un nombre minimum des états, cela revient à rechercher sur les états border et notifier celui qui comporte les valeurs minimales pour les paramètres (β, γ). La recherche est focalisée sur les valeurs local généré par la machine. Ce minimum est en faveur de la machine sur laquelle les états ont été importé pour compenser le déséquilibre, de plus il n’y a pas d’éléments commun entre ensemble de dépendance du minimum et celle des états à importer car cela évitera de réaliser une double itération. La formalisation de ce principe est la suivante :

 ;

Pour alors

Si alors

Pour alors

Si (( )or (x and ( )) and( ==null or (( )or ( and ))) alors

;

Finsi

finpour

Sinon

Site= findById;

Si ( !=null) and (site !=null) and((MF==null) or (MF.β>x.β) or (MF.β==x.β and MF.γ>x.γ)) and ((x.β<site.β) or (x.β==site.β and x.γ<site.γ)) alors

;  
 finsi

finpour

Après avoir retrouver des valeurs minimums à importer ou déplacer, avant l’envoi des états on vérifie si nombre d’état à déplacer et à importer d’entraine pas un déséquilibre, lorsque le déséquilibre est toléré, on redéploye les états concernés, sinon ont vérifié entre les actions (déplacer ou importer) celui qui peut être réaliser et on l’applique. Celui-ci peut être formalise comme suit :

Si | alors

Sinon si alors

Sinon si

Finsi

Ainsi à la réception soit on ajouter les états envoyés ou on envoie les états demander, après cela on reprend la procédure de recherche de minimum afin de vérifié s’il est possible de réaliser des redéploiements.

Réception (elementsj, MGi) par la machine j

Si alors

ajouter

finsi

si

finisi

Reprendre l’étape 2.

Cette fonction recherche les états a envoie.

Lorsqu’on effectue une vérification AG(a) sur la structure distribué (Figure 9) les résultat obtenu sont :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **T** | **VL** | **NR** | **RF** | **M** | **ME** | **Id** | **T** | **VL** | **NR** | **RF** | **M** | **ME** | **Id** | **T** | **VL** | **NR** | **RF** | **M** | **ME** |
| S1 | I | 0 | 2 | C | M1 |  | S9 | I | 0 | 1 | E | M2 |  | S15 | I | 0 | 1 | C | M3 |  |
| S2 | I | 0 | 2 | C | M1 |  | S7 | E | 0 | 3 | C | M2 | M1 | S16 | I | 0 | 1 | C | M3 |  |
| S3 | I | 0 | 2 | C | M1 |  | S11 | I | 0 | 4 | C | M2 |  | S17 | I | 0 | 1 | C | M3 |  |
| S5 | I | 0 | 2 | C | M1 |  | S10 | I | 0 | 1 | C | M2 |  | S18 | I | 0 | 1 | C | M3 |  |
| S6 | I | 0 | 2 | C | M1 |  | S12 | I | 0 | 1 | C | M2 |  | S19 | I | 0 | 1 | E | M3 |  |
| S7 | I | 0 | 2 | C | M1 |  | S14 | I | 0 | 1 | E | M2 |  | S20 | I | 0 | 3 | C | M3 |  |
| S8 | I | 0 | 2 | C | M1 |  | S15 | E | 0 | 2 | C | M2 | M3 | S21 | I | 1 | 1 |  | M3 |  |
| S9 | E | 0 | 2 | E | M1 | M2 | S13 | I | 0 | 4 | C | M2 |  | S2 | I | 0 | 3 | C | M3 | M1 |
| S10 | E | -1 | 2 | C | M1 | M2 | S22 | E | 0 | 4 | C | M2 | M3 | S23 | I | 0 | 3 | C | M3 |  |
| S11 | E | -1 | 5 | C | M1 | M2 |  |  |  |  |  |  |  | S11 | E | 0 | 5 | C | M3 | M2 |

D’après le tableau précèdent on constate qu’il a eu beaucoup d’échange entrer m1, m2 et m2, m3, leur minimisation entrain une diminution du temps de traitement. Le parcourt de minimisation (α, β, γ) entre m1 et m2 se résume dans le tableau ci-dessous :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **M2** | | | | | | |
| **M1** |  | **S9** | | **S10** | | **S11** | |
| **T** | α1, β2+7, γ2 | α1, β1, γ2+1 |  |  |  |  |
| **T** |  |  | α1, β2+5, γ2 | α1, β1+3, γ1+1 |  |  |
| **T** |  |  |  |  | α1, β2+2, γ2 | α1, β1+2, γ1 |
|  | **T** | | **T** | | **T** | |
| **S7** | α2, β2+7, γ2 | α2, β2+1, γ1 |  |  |  |  |

Tableau 13

En se basant sur l’ordre des échanges avec l’état s7, l’exécution de la stratégie de m1 entrain un déséquilibre alors m2 peut effectuer un déplacement de ces états vers m1 et avec s11 m1 effectue déplacement. L’exécution de ces 2 action empêche une 5eme itérations et 3eme également entre m1 et m2. Cette exécution supprime les stratégies avec s9, il ne reste plus que s10 ou on peut faire qu’un équilibrage de charge vue la différence de nombre entrer les machines. La redistribution obtenue est la suivante:

a

a

a

a

b

a

S3

S5

S8

S9

S6

S7

a

a

┴

a

a

b

┴

a

a

S1

S2

S11

S13

S22

S15

S14

S12

S10

┴

S10

┴

S3

Après la redistribution il ne reste plus que des relations entre m1 et m2, m2 et m3.

En appliquant le même principe de minimisation pour m2 et m3, le résultat est représenté dans le tableau ci-dessous.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **M3** | | | | |
| **M2** |  | **S15** | | **S22** | |
| **T** | α1, β2+3, γ2 | α1, β1+4, γ1+1 |  |  |
| **T** |  |  | α1, β2+3, γ2 | α1, β1+1, γ1 |
|  | **T** | | **T** | |
| **S2** | α2, β1+3, γ2 | α2, β1+2, γ1 |  |  |
| **S11** |  |  | α2, β2+2, γ2 | α2, β1+3, γ1 |

Tableau 14

La stratégie qui avantage les deux machines est le fait que s2 n’appartient à qu’aucun des deux, dans ce cas la machine m3 déplace les états de s22 ce qui réduit l’itération 4 et 3 dans les 2 machines, la suite des stratégie entrain un déséquilibre.

Le résultat de la redistribution est la suivante :

a

a

a

a

b

a

S3

S5

S8

S9

S6

S7

a

a

a

┴

a

b

┴

a

S1

S2

S11

S13

S22

S15

S14

S12

a

S10

┴

S12

a

a

a

a

b

a

┴

a

┴

S15

S17

S19

S21

S2

S20

S11

S18

S16

Avec cette configuration on passe de 5 itération à 3 qui est la plus optimal.

Le choix d’une stratégie repose sur la coopération et la relation des états afin de redirigé les états dans la machine la plus approprié.

* 1. **Stratégie**

Le choix des stratégies repose sur l’ordre décroissant des itérations, la minimisation repose sur éléments suivantes :

* Soit les éléments d’une stratégie S= {s1, s2, s3},
* Soit les éléments d’une itération I= {i1, i2, i3},
* Soit les stratégies indépendantes SI= {(si, sj)/ si, sj ɛ S²,si∩sj=Ø,
* Soit les stratégies dépendantes dans l’ordre croissante SD= {si, sj, sk},
* Les ensembles de SI peuvent être exécuté sans risque s’il garantit un bon équilibre,
* Il peut y avoir un compromis entre SI et SD pour minimiser une itération,
* L’ensemble SD peut être utiliser pour équilibre la charge

# Algorithme de distribution d’espace d’états basée sur le comportement des systèmes

Dans cette partie on propose une nouvelle politique de distribution qui vise à analyser le comportement du système, pendant l’exécution du modèle checker sur les espaces d’états générer. Cette politique vise à établir un rapport à partir de l’exécution, et redistribue les états qui réduisent le temps de réponse et améliore les performances du système. En donnant une meilleure localité à ces états et respect l’équilibrage des espaces d’états entre les machines.

Le rapport élaboré comporte les statistiques de chaque état, celui-ci est le comportement des états par rapport à la vérification d’une formule. Le protocole effectue une étude à base des statistiques pour migrer, dupliquer ou laisser un ensemble d’états.

Dans la suite du travail nous appliquons les phases de ce politique sur un graphe distribue.

## Génération de l’espace d’état distribue

Une machine choisie aléatoirement, appelée "initiatrice", commence le processus de la génération de l’espace d’états, par l’exploration de l’état initial en générant tous ses états successeurs. Par la suite, toutes les machines disponibles sur le réseau contribuent itérativement à la construction des fragments de l’espace d’états distribué.

Pour chaque nouvel état généré appartenant à une machine Mi, tous ses états successeurs sont générés. Un état successeur peut être dans la même machine ou dans une machine distante. Chaque machine Mi envoie tous ses états externes aux machines s’en chargeant du traitement de cet état tout en indiquant que Mi est un *subscriber* de cet état.

**Exemple**

┴

┴

b

a

S1

S2

S3

S4

**Machine : M1**

S4

┴

b,c

S2

**Machine : M2**

S5

S3

S4

┴

┴

c

a,b,c

S1

**Machine : M3**

Figure 10 Distribution des espaces d'états

## Construction des statistiques et des rapports

Apres la génération on obtient un espace d’états réparti du système, où le nombre des fragments disjoints est égal au nombre de machines disponibles sur le réseau. Cependant, cette distribution n’assure pas une meilleure performance lors de l’exécution du modèle checker, due par une distribution les états fortement lié, cela engendre assez calcule pour détecter la valeur logique de cet ensemble états. En effet la détection de celui-ci est réalisable grâce au rapport générer lors de l’exécution du modèle checker.

La génération du rapport en utilisant la théorie de kleene est réalisé comme suit : après une itération, lorsque la valeur logique des états dite *Publisher* est fausse, les machines les sollicitant sont notifier et prend en compte ces notifications pour effectuer une nouvelle itération. La construction est incrémental pour chaque itérations du point fixe effectué c’est-à-dire un calcule engendre par la première itération ou une notification causé par un changement d’état externe. Ainsi un état est constitué de :

* **Id** : identifiant
* **T** : Type d’état (I : interne ; E : externe)
* **P** : Un ensemble de propriétés
* **LS** : Liste de *subscribers*
* **ME** : Machine stockant l’état Externe
* **UF** : ensemble de formules vérifié
* **VL** : valeur logique de la formule a vérifié
* **RF**: raison pour laquelle VL est faux (c : chemin/e : état)
* **MF**: Machine qui pose plus de problème
* **NR** : nombre de recalcule permettant de déterminer la valeur logique

**Exemple** de rapport AG (aVc)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | T | VL | NR | RF | M | ME |
| S1 | I | 0 | 1 | E | M1 |  |
| S2 | I | -1 | 1 |  | M1 |  |
| S3 | E | -1 | 1 |  | M1 | M3 |
| S4 | E | -1 | 1 |  | M1 | M2 |
| S2 | E | -1 | 1 |  | M2 | M1 |
| S4 | I | -1 | 1 |  | M2 |  |
| S4 | E | -1 | 1 |  | M3 | M2 |
| S5 | I | 0 | 2 |  | M3 |  |
| S3 | I | -1 | 1 |  | M3 |  |
| S1 | E | 0 | 2 | E | M3 | M1 |

Table 1 Statistique des états

## Politique de redistribution

La politique de redistribution est un processus d’optimisation basé sur les statistiques, permettant de regrouper les états fortement lié dans une machine si possible sinon réduire le temps de traitement. La bonne répartition revient de trouver les états à partir des quels peux de calculs est engendré. Ainsi principe sont utiliser pour aboutir à une bonne partition : la migration ou la duplication des états.

1. **Duplication**

Lorsque la valeur logique d’état est faux causé par l’état lui-même c’est-à-dire formule n’est pas vérifier pour cette état mais vérifier pour tous ces successeurs, dans ce cas précise, la duplication de cette état dans les machines qui le sollicite permet de réduire le temps de recalcule, car lorsque la machine dispose une copie de l’état, la valeur logique est déduit directement, ce qui empêche une notification pour déduire celui-ci.

1. **Migration**

Lorsque la valeur logique d’un état (subscriber) dépend de ces successeurs, alors la distribution de ces états fait croitre le temps de traitement, pour résoudre ce problème, on cherche une meilleur localité pour ces états qui vise à garantir un temps de trament minimum et un bon équilibrage. La mise en œuvre de ce mécanisme est la suivante :

* Apres la réception des notifications coté Subscriber

Tous les états possédant des successeurs externes avec une valeur logique fausse due à un problème de chemin, alors on cherche à effectuer une partition de sorte que tous les états prédécesseur affecté par ce changement devront être migrés y compris les successeurs présentant des valeurs fausses liées à des problèmes de chemin. Ainsi cette demande est envoyer vers la machine la plus sollicité si la migration de ces états m’empêche pas d’avoir au moins 20 % de l’espace d’états global, sinon les machines sont sollicité de migrer vers celui-ci si possible.

* Apres la réception des notifications coté Publisher

Une machine qui est sollicité, peut aussi solliciter d’autre machine, alors après avoir reçu les notifications nécessaires, il peut entamer le processus. Il peut aussi recevoir des suggestions d’un ensemble d’espace d’états ou des demandes d’envoi des espaces d’états.

Ainsi les états (Publisher) ayant une valeur logique fausse due par des chemins des successeurs soit par des états externe ou par des états interne de la machine, alors on cherche la limite (par rapport aux états interne) à partir du quels on ne dispose pas de problème de chemin. Cette limite concerne les successeurs et les prédécesseurs. Apres avoir cette limite on procède comme suit :

* S’il reste suffisamment d’espace d’états et aucune suggestion n’est reçu alors la machine la plus sollicité est susceptible d’accueillir ces états.
* Si on a une suggestion et des états externes sont impliquer et il restant suffisamment d’espace d’états alors on choisit une machine qui sollicite, diffèrent de celui qui suggère, pour prendre en charge ces états.
* Sinon s’il ne reste pas assez d’états pour réaliser la migration, s’il peut accueillir les suggestions il accepte.
* Sinon il trouve une bonne localité pour ces états

Ces 4 points vise à partager les états pour garantir un minimum de calcule pour réaliser le traitement.

**Exemple** rapport : AG (aVc) après la redistribution

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | T | VL | NR | RF | Machine | ME |
| S1 | I | 0 | 1 | E | M1 |  |
| S2 | I | -1 | 1 |  | M1 |  |
| S3 | E | -1 | 1 |  | M1 | M3 |
| S4 | E | -1 | 1 |  | M1 | M2 |
| S2 | E | -1 | 1 |  | M2 | M1 |
| S4 | I | -1 | 1 |  | M2 |  |
| S4 | E | -1 | 1 |  | M3 | M2 |
| S5 | I | 0 | 1 |  | M3 |  |
| S3 | I | -1 | 1 |  | M3 |  |
| S1 | E | 0 | 1 | E | M3 | M1 |

En appliquant le protocole le mécanisme de duplication de l’état S1 dans la machine3 permet de réduire le temps de traitement et de réaliser une seul itération et conclure que la formule n’est pas vérifiée.

Table 2 Statistique des espaces d'états

Table 3 Redistribution des espaces d'états

┴

┴

b

a

S1

S2

S3

S4

**Machine : M1**

S4

┴

b,c

S2

**Machine : M2**

S5

S3

S4

b

┴

c

a,b,c

S1

**Machine : M3**

**[BICHOT, 2007]** Charles-Edmond BICHOT (2007). Élaboration d’une nouvelle méta heuristique pour le partitionnement de graphe : la méthode de fusion-fission. Application au découpage de l’espace aérien. Thèse de doctorat, TOULOUSE.

**[Bouneb, 2011]** BOUNEB, Z. E.-A. (2011). Vériﬁcation Symbolique des Systèmes Critiques: Approche Distribuée. Thèse de doctorat, Université de Mentouri, Constantine, Algeria.

**[François Queyroi, 2013]** François Queyroi. Partitionnement de grands graphes : mesures, algorithmes et visualisation. Autre [cs.OH]. Université Sciences et Technologies - Bordeaux I, 2013. Français.

**[Deo, 2017]** DEO, N. (2017). Graph theory with applications to engineering and computer science. Courier Dover Publications.

##### [**EULER**] EULER, “Solutio Problematis ad Geometriam Situs Pertinentis”. Commentarii Academiae Scientarum Imperialis Petropolitanae 8, 128-140, 1736

**[J&O, 1944]** J. Von Neumann, O. Morgenstern. “Theory of Game and Economic Behavior”. Princeton University Press. 1944.

**[**[**John von Neumann**](https://fr.wikipedia.org/wiki/John_von_Neumann)**, 1928]** [John von Neumann](https://fr.wikipedia.org/wiki/John_von_Neumann), « Zur Theorie der Gesellschaftsspiele », [Mathematische Annalen](https://fr.wikipedia.org/wiki/Mathematische_Annalen" \o "Mathematische Annalen), vol. 100, no 1,‎ 1928, p. 295-320

**[**[**Ernst Zermelo**](https://fr.wikipedia.org/wiki/Ernst_Zermelo)**, 1913]** [Ernst Zermelo](https://fr.wikipedia.org/wiki/Ernst_Zermelo), « Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels », Proceedings of the Fifth International Congress of Mathematicians,‎ 1913

**[Norman]** (en) Norman L. Biggs, *Discrete Mathematics*, Oxford University Press ([ISBN](https://fr.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-19-850717-8](https://fr.wikipedia.org/wiki/Sp%C3%A9cial:Ouvrages_de_r%C3%A9f%C3%A9rence/978-0-19-850717-8))

**[**[**Émile Borel**](https://fr.wikipedia.org/wiki/%C3%89mile_Borel)**, 1921]** [Émile Borel](https://fr.wikipedia.org/wiki/%C3%89mile_Borel), « La théorie du jeu et les équations intégrales à noyau symétrique », [Comptes rendus hebdomadaires des séances de l'Académie des sciences](https://fr.wikipedia.org/wiki/Comptes_rendus_hebdomadaires_des_s%C3%A9ances_de_l%27Acad%C3%A9mie_des_sciences), no 173,‎ décembre 1921

**[Nash, 1950]** [*John Nash*](https://fr.wikipedia.org/wiki/John_Forbes_Nash)*, «*Equilibrium points in n-person games*»,*[PNAS](https://fr.wikipedia.org/wiki/Proceedings_of_the_National_Academy_of_Sciences)*, vol. 36, no 1,‎ 1950, p. 48-49*

**[Nash, 1950]** *John Nash, «*The Bargaining Problem*»,*[Econometrica](https://fr.wikipedia.org/wiki/Econometrica" \o "Econometrica)*, vol. 18,‎ 1950, p. 155-162*

**[Nash, 1951]** *John Nash, «Non-cooperative games »,[Annals of Mathematics](https://fr.wikipedia.org/wiki/Annals_of_Mathematics" \o "Annals of Mathematics) ,vol . 54 ,‎1951 ,p.286–295*

[Article1] [Apprentissage automatique](https://www.ionos.fr/digitalguide/web-marketing/analyse-web/quest-ce-que-lapprentissage-automatique/) , http://dictionnaire.sensagent.leparisien.fr/Apprentissage%20automatique/fr-fr/#cite\_note-2

[Article2] Q-learning, https://en.wikipedia.org/wiki/Q-learning

,