Rapport de Stage
Découverte Distribuée de Nouvelles Conséquences
–
Génération de problème et étude expérimentale

Remy Bin
sous la supervision de Gauvin Bourgne

September 29, 2015

1 Introduction

Le but du présent document est de résumer le travail effectué au département ACASIA durant les deux mois de l'été 2015.

Pour limiter la verbosité, nous nous contenterons de résumer ici les différentes activités et thèmes abordés, en donnant des pointeurs vers les annexes présentes avec ce document pour des explications plus approfondies.

La structure de la présentation suivra l'ordre chronologique. Cela permettra de rendre compte des problématiques rencontrées au fur et à mesure de l'avancement du projet et des dépendances des différentes parties entre elles.

1.1 But du stage

Voici le sujet du stage de Rémy Bin, dans le cadre du projet LIP6 DCSMA (Découverte de Conséquences dans un Système Multi Agents).

Intitulé du stage : Découverte Distribuée de Nouvelles Conséquences - Génération de problème et étude expérimentale

Sujet: Ce stage s'inscrit dans le cadre de la découverte incrémentale de conséquences dans un système multi agent. Etant donné un ensemble d'agent ayant chacun une théorie clausale et un ensemble d'observations arrivant dans le système, il s'agit de permettre aux agents d'inférer toutes les nouvelles conséquences de ces observations respectant une certain biais de langage. Des protocoles ont été proposés pour permettre aux agents de se communiquer les informations nécessaires au raisonnement [2]. Il a été constaté que dans certains cas ces protocoles permettent d'économiser des opérations, mais que dans d'autres, le nombre de communications et d'opération à effectuer explose. Le but de ce stage est de réaliser une étude expérimentale poussée avant de déterminer les conditions d'utilisation de l'algorithme de base et de ces variantes. En l'absence de base de problèmes classiques dans ce domaine, il faudra tout d'abord générer automatiquement un certain nombre de problèmes, en particulier en adaptant des problèmes de preuves automatiques (TPTP ou SAT). Il s'agit de transformer ces problèmes en proposant notamment un biais de langage, un ensemble d'observations et une distribution de la théorie, de réaliser un script pour tester l'algorithme sur les problèmes générés et d'analyser les résultats. Par la suite, on réalisera une étude plus poussée en formalisant des problèmes de dilemmes éthiques.

2 Chronologie

2.1 Prise en main du système

Après avoir consulté la bibliographie fournie par mon encadrant : [3], [4] et [1], j'ai repris le code et le rapport du précédent stagiaire, qui s'appuyait déjà sur les plateformes décrites dans ces articles. Les rappels théoriques peuvent être trouvés dans *pedro rapport de stage filename*

Il a fallu dans un premier temps mettre en place l'environnement de développement nécessaire et résoudre les conflits existants dans le projet en identifiant les changements apportées ainsi que les modules dont j'aurai besoin plus tard.

En général, chaque module a été défini comme une boîte noire avec des entrées et des sorties, et si quelques tests invalidaient cette définition, il fallait alors se plonger dans le code légué. Opération non triviale. Le fichier *put name here* récapitule les arguments nécessaires pour chaque module ainsi que leurs fonctionnalités.

2.2 Expérimentations

2.2.1 Première expérience

Une fois les fonctionnalités des modules définies, un premier lanceur d'expérience simple a été construit (en python). *put name exécutable here*

Il permettait simplement de lancer une expérience unique en générant les multiples dépendances implicites du lancement d'un problème de découverte distribuée.

2.2.2 Ajout de fonctionnalités

Ensuite, des fonctionnalités ont été ajoutées au système initial (Java) pour: -récupérer la liste des conséquences trouvées dans un fichier externe *put name class here* -produire les comparaisons entre deux fichiers de conséquences (utile pour comparer les performances d'un processus interrompu par rapport au problème résolu) *put name class here* Et dans le nouveau système, un module de paritionnement des Observations (entre Top-clause et Axioms) a été réalisé.

2.2.3 Parallèlisation

Le but du sytème était de pouvoir lancer un grand nombre d'expériences en faisant varier les paramètres et à partir des données produites, de vérifier des hypothèses et d'en générer de nouvelles. Une fois le processus d'une expérience défini, il a donc fallu créer un système pour lancer de multiples expériences sur plusieurs machines (cluster Big du LIP6). Dans le fichier *put name file here* sont récapitulées les instructions de base nécessaires pour le lancement des expériences. Celles-ci sont récapitulées lors du lancement de chaque expérience.

En s'aidant du module Multiprocessing de python, la parallèlisation des expériences a ensuite été abordée. Ne pouvant savoir tout à fait ce dont nous pourrions avoir besoin par la suite, il été décidé de sauvegarder tout ce qui était possible pour pouvoir en tirer de l'information a posteriori. Ainsi, les sorties de tous les modules ont été loggées.

Le premier problème sur lequel le système avait été testé était un problème de bio : glucolysis. C'est sur ce premier problème que nous avons lancé les premières expériences de découverte incrémentale pour mono-agent. Cela nous permettant d'avoir une bonne vision des problèmes difficiles. À partir des

problèmes finissant, nous lancerions les problèmes multi-agents correspondants pour pouvoir les comparer à leur version mono-agent.

2.2.4 Génération de Problèmes

Pendant que les expériences tournaient sur le cluster, nous nous sommes procurés des problèmes de Thorem Proving (TPTP) et de Satisfiabilité (cs.ubc.ca). L'idée était de générer de nouveaux types de problèmes pour tester nos hypothèses sur des jeux de données plus grand.

Il a fallu filtrer chacun des problèmes selon ses spécificités : -ne garder que les problèmes satisfiables -essayer de garder des problèmes correspondant à des chosses concrètes etc... (dans le futur, il faudrait aussi faire une fonction pour rendre plus explicite des théories en remplacant les noms des prédicats par des choses concrètes) -etc...

Certains problèmes étant en outre assez longs et potentiellemt difficilement finissables, après conversion dans le formalisme logique correspondant, il a été opéré une division de ces théories, en se soumettant à la contrainte d'avoir toujours un nombre suffisant de clauses dans la théorie produite pour pouvoir répartir un certain nombre de clauses entre un nombre maximal d'agents.

2.2.5 Analyses

À ce stade, les problèmes monoagent étaient finis, et nous avons alors lancé la résolution des problèmes issus des nouvelles données pour du monoagent. En sélectionnant les problèmes monoagent de BIO, nous avons aussi lancé la résolution en multiagent pour les différentes variantes d'algorithme et de paritionnement entre agents.

Avec la fin des premières expériences, nous avons procédé aux première analyses. Nous nous sommes efforcés d'identifier de potentielles anomalies et incohérences et certains comportements inattendus se sont fait jour. [Rq : toutes les analyses détaillées se trouvent dans le fichier *put filename here notebook* joint]

2.2.6 Réusinage du système

Malgré le nombre important de paramètres testés, il restait à raffiner de nombreuses hypothèses, et il fallait lancer de nouvelles expériences pour les tester. De plus, toutes les expériences n'étaient très pertinentes : parfois redondantes. Or les ressources computationelles étant limitées, nous nous sommes fait la réflexion qu'il valait mieux orienter la conception du système vers quelque chose qui forcerait la formulation et la vérification d'hypothèse plutôt que d'exploser combinatoirement sans direction et d'utiliser la force brute des données, trop coûteuses à générer. Chaque expérience prenait un temps considérable et nécessitait un certain nombre de ressources qui n'étaient pas toujours disponibles au moment opportun, ces contraintes orientaient aussi notre conception vers un système facilitant les itérations plus petites et plus nombreuses (permettant une meilleure supervision des hypothèses et forcant leur formulation si déficientes).

L'interruption d'expérience n'était pas un problème grâce à des logs précis de ce qui devait être fait et de ce qui avait été fait, il suffisait ainsi de filtrer des listes. Le problème était des expériences parallèleles, "en parallèle".

Les expériences étant toutes lancées en parallèle à partir de de processus initiateurs différents, nous n'avions pas prévu les cas de ressources partagées à ce niveau et nous avons alors décidé de dupliquer les répertoires sources des problèmes pour y faire tourner chaque expérience indépendante pour ne pas avoir de conflits.

Cette solution n'était évitablement pas viable à long terme et était favorable à l'apparition de doublons ainsi qu'à la complexification de l'arborescence des fichiers résultats.

Il fallait que toutes les expériences puissent avoir lieu dans le même répertoire et qu'il n'y ait pas de conflits, tout en identifiant sans ambiguité la source des données générées (à quel moment, pour quelle hypothèse). Pour ce faire, nous avons proposé un système donnant un nom à chaque expérience. Chaque expérience est loggé dans un fichier récapitulant les conditions exactes lors du lancement, et avec quels paramètres. Certaines expériences peuvent être indépendeantes et d'autres incrmentales (se baser sur des expériences précédentes). De même, pour contrôler le comportement des modules externes utilisés, nous avons surchargé les options de remplacement de fichier. Pour avoir une vision d'ensemble des expériences en cours, nous avons mis en place un système de Logging particulier (à partir du module Loggin de python), où les évènements peu fréquents sont notés par des exceptions. Chaque expérience atomique (une découverte) a son log exhaustif et les logs importants sont "remontés" à un log global pour toute l'expérience. Ainsi pour suivre l'état de toute expérience à tout moment, il suffit de lire le log de l'expérience globale.

Les autres propositions du système sont détaillées dans les fichiers de documentation fournis avec le code.

3 Conclusion

À partir d'un système complexe, les efforts effectués lors de ce stage ont mené à la production d'un système et d'une architecture suffisamment abstraite pour pouvoir être reprise à l'occasion d'une autre étude. Nombre des outils seront ainsi suffisamment documentés et testés pour permettre leur réutilisation par un utilisateur nouveau.

References

[1] Gauvain Bourgne and Katsumi Inoue. Partition-based consequence finding. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, pages 641–648, 2011.

[2] Gauvain Bourgne and Nicolas Maudet. Finding new consequences of an observation in a system of agents (Extended Abstract). (June):1223–1224, 2012.

[3] Katsumi Inoue, Koji Iwanuma, and Hidetomo Nabeshima. Consequence finding and computing answers with defaults. *Journal of Intelligent Information Systems*, 26(1):41–58, 2006.

[4] Hidetomo Nabeshima, Koji Iwanuma, Katsumi Inoue, and Oliver Ray. SOLAR: An automated deduction system for consequence finding. *AI Communications*, 23(2-3):183–203, 2010.