Chapter 2

Résumé en Français

2.1 Introduction

Actuellement, les responsables de la protection contre les situations de désastre doivent prendre les décisions sur la préparation et l'exécution des plans d'évacuation en considérant les causes potentielles du désastre. Pour cette raison, est sería souhaitable de développer un système capable d'obtenir et d'analyser les plans d'évacuation. Où ce système est basé sur la connaissance relative a una region particuler, des données géographiques et de leurs propres capacités. En plus, le système doit être capable d'échanger l'information et les services avec des systèmes analogues, ainsi que avec les êtres humains.

basés sur la connaissance du milieu particulier, des données géographiques et de leurs propres capacités et d'échanger l'information et les services avec des systèmes semblables comme pour les êtres humains. Pour développer un tel système, on peut utiliser Answer Set Programming (ASP). ASP est un langage de programmation logique et déclaratif pour la représentation de la connaissance [14]. ASP représente un nouveau paradigme pour la représentation logique qui permet, selon le concept de la négation par échec, de résoudre des problèmes avec la connaissance par défaut et de permettre

un raisonnement non- monotone.

L'objectif de notre travail est précisément *de faire des recherches et d'évaluer les capacités de ASP à représenter des situations de désastre ayant pour objectif d'aider à définir des plans d'évacuation.* La motivation de notre travail se base sur l'idée que ASP possède de nombreuses qualités qu'un tel programme doit avoir: Il est possible de traduire l'information géographique dans un format que ASP peut comprendre. Il existe *Answer Set Planning* qui procure une manière naturelle et élégante de modeler les problèmes de planification [16]. ASP utilise le concept de négation par échec qui permet d'exprimer des exceptions, des restrictions et de représenter une connaissance incomplète. Il existe en plus, dans ASP, plusieurs approches afin d'exprimer des préférences.

Pour atteindre notre objectif, nous commençons par l'étude du format de l'information géographique. En nous basant sur notre propre expérience, nous proposons un procédé pour extraire la connaissance nécessaire afin de représenter la zone de danger de l'in-formation géographique. Nous considérons aussi que, normalement, dans une zone à risque, il y a un ensemble de voies d'évacuation prédéfinies. Pourtant, dans une situation réelle, il est possible qu'une partie de ces voies prédéfinies soient bloquées. Dans ce cas, la définition de plans alternatifs d'évacuation est nécessaire. Nous utilisons *Consistency restoring rules* (CR-régles) pour obtenir les plans d'évacuation alternatifs. Nous montrons également que les programmes avec CR-régles peuvent se représenter correctement en utilisant des programmes de disjonction ordonnée (ODLP).

Nous proposons d'utiliser le langage $\mathcal{PP}$ pour exprimer des préférences à divers niveaux sur les plans alternatifs. Nous définissons le langage $\mathcal{PP}^{par}$, comme extension du langage $\mathcal{PP}$ où ses connectifs permettent que nous représentions de manière compacte les préférences qui ont une caractéristique particulière. En plus, nous présentons une brève description au sujet de la relation entre le langage $\mathcal{PP}$ et la logique linéaire

propositionnelle temporelle (LTL), puisque nous considérons que le langage $\mathcal{PP}$ peut profiter de travail deja fait pour LTL pour exprimer des préférences.

Nous proposons aussi une extension de ODLP à une classe plus vaste de programmes logiques. Il est important de mentionner qu'un sous-ensemble de ces programmes est utile pour permettre une codification plus simple et plus facile afin d'obtenir les plans préférés respectivement aux préférences exprimées dans $\mathcal{PP}$. Finalement, nous introduisons la notion de *Contenu Sémantique d'un programme* comme point de vue alternatif pour obtenir des variantes de la sémantique de Answer Sets. Une de ces variantes est une sémantique nouvelle introduite dans ce travail appelée *answer sets parciels.*

2.2 Etat de l' art

2.2.1 Answer Set Programming

L'utilisation de Answer Set Programming (ASP) permet de décrire un problème informatique comme un programme logique auquel answer sets procure les solutions du problème donné. Dans ce travail, *un programme est interprété comme une théorie propositionnelle* et l'unique négation utilisée est la *négation par échec.* Pour les lecteurs non familiarisés avec cette approche, nous recommandons [38, 33]. Donc, notre propos se limite aux programmes propositionnels.

Nous utiliserons *le langage de logique propositionnel* de la manière habituelle, où les symboles propositionnels sont: $p, q, \ldots$; les connectifs propositionnels sont: $\wedge, \vee, \rightarrow, \bot$, et les symboles auxiliaires sont: $(,)$. Une *formule propositionnelle bien formée* est définie inductivement comme suit: Un symbole propositionnel est une formule propositionnelle bien formée. Si f et g sont des formules propositionnelles bien formées, alors $\neg f$, $f \wedge g$, $f \vee g$, $f \rightarrow g$ le sont aussi. Une formule propositionnelle bien formée peut être appelée

seulement *formule*. Un *atome* est un symbole propositionnel. Pour toute formule donnée bien formée f, nous assumons que $\neg f$ est exactement l'abréviation de $f \rightarrow \bot$ y $\top$ qui est l'abréviation de $\bot \rightarrow \bot$. Nous insistons sur le fait que l'unique négation utilisée dans ce travail est la *négation par échec* et elle est représentée par le symbole $\neg$. Il vaut la peine de mentionner qu'il est toujours possible de traiter l'autre négation appelée *classique* ou encore *négation forte*, rendue par $-$, en transformant les atomes avec négation classique [15]. Chaque atome avec négation classique, $-a$, qui apparaît dans la formule est remplacé par un nouvel atome, a°, et la règle $\neg(a \wedge a^\circ)$ est ajoutée. La règle $\neg(a \wedge a^\circ)$ peut aussi être écrite comme $(a \wedge a^\circ) \rightarrow \bot$, basé sur le fait que $\neg f$ est juste l'abréviation de $f \rightarrow \bot$.

Une *signature* $\mathcal{L}$ est un ensemble fini d'atomes. La *signature d'un programme* P, notée comme $\mathcal{L}_P$, est un ensemble d'atomes qui apparaissent dans P. Une *littéral* est n'importe quel atome p (un littéral positive) ou la négation d'un atome (un littéral négatif). Une *littéral nié* est le signe de négation $\neg$ suivi par un littéral, i.e. $\neg p$ or $\neg\neg p$. En particulier, $f \rightarrow \bot$ est appelé *restriction* et elle est aussi indiqué comme $\leftarrow f$. Une *théorie régulière* ou *programme logique* est uniquement un ensemble de formules bien formées, et peut appeler *théorie* ou *programme* quand ils ne surviennent pas d'ambiguïtés.

Dans quelques définitions, nous utilisons *logique intuitionniste*, lequel sera signaler par I. Pour un ensemble donné d'atomes M et un programme P, nous écrirons $P \vdash_{\mathrm{I}} M$ pour abréger $P \vdash_{\mathrm{I}} a$ pour toute $a \in M$. Pour un ensemble donné d'atomes M et un programme P, nous écrirons $P \vdash_{\mathrm{I}} M$ pour indiquer que: $P \vdash_{\mathrm{I}} M$, et P est consistant relativement à I (c'est à dire qu'il n'y a pas de formule A tel que $P \vdash_{\mathrm{I}} A$ et $P \vdash_{\mathrm{I}} \neg A$). Parfois, nous écrirons $\vdash$ au lieu de $\vdash_{\mathrm{I}}$ quand ils ne surviennent pas d'ambiguïtés. Définissons maintenant les answer sets (ou modèles stables) des programmes logiques. La sémantique de answer sets a été d'abord définie dans les termes de *réduction Gelfond-*

Lifschitz [14] et elle est habituellement étudiée dans le contexte des transformations dépendantes de la syntaxe dans le programme. Nous suivons une approche alternative étudiée par Pearce [38] et par Osorio et.al. [33]. Cette approche caractérise les answer sets d'une théorie propositionnelle en terme de logique intuitionniste et elle est présentée dans le théorème suivant. La notation est basée sur [33].

Theorem 2.1. *Etant donné P une théorie et M un ensemble d'atomes. M est un answer set pour P si et seulement si $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$.*

Comme annotation additionnelle utilisée dans le reste de ce travail, on trouvera: Un ensemble fini de formules P est *consistant* [27] s'il n'y a aucune formule A tel que A et $\neg A$ soient des théorèmes de P. Un ensemble fini de formules P est *complet* [27] si, pour n'importe quelle formule A de P, $P \vdash A$ ou $P \vdash \neg A$. Un ensemble fini de formules P' est une *extension* d'un ensemble fini de formules P [27] si chaque théorème de P il y a un théorème de P'. Etant donné deux ensemble X et Y, nous écrivons $X \subset Y$ pour indiquer que $X \subseteq Y$ and $X \neq Y$.

Comme d'habitude dans Answer Sets Programming, nous admettons qu'un programme avec des symboles du prédicat est seulement une abréviation du programme ground, indiqué comme $ground(P)$.

Dans certains cas, nous avons besoin de modeler quelques problèmes en utilisant des symboles du prédicat avec des variables, ces variables doivent être présentées uniquement avec un sous-ensemble de l'univers de Herbrand du programme. Ce type de programme s'appelle *programmes pourvu de classes avec des symboles du prédicat* [4]. Nous les utiliserons par exemple quand nous étudierons la codification d'un problème de planification dans la Section 2.2.4. On trouve actuellement plusieurs résolutions de answer sets. Dans ce travail, nous avons utilisé: DLV[1] , SMODELS[2] , un "frontend"

[1] http://www.dbai.tuwien.ac.at/proj/dlv/
[2] http://www.tcs.hut.fi/Software/smodels/

pour DLV appelé DLVK[3] , une modification de SMODELS qui peut être utilisée pour obtenir les answer sets préférés selon la sémantique de disjonction ordonnée appelée PSMODELS et le Logics Workbench(LWB)[4] qui nous permet de travailler avec la logique intuitionniste.

2.2.2 CR-Programmes et Programmes Abductifs

Les règles pour restaurer la consistance (CR-règles) sont les règles qui s'ajoutent aux programmes logiques alternatifs standard, mais ils s'appliquent seulement quand les règles standard dans le programme mènent a l'inconsistance. Une CR-règle est écrite comme $\alpha \overset{+}{\leftarrow} \beta$, ce qu'on lit intuitivement comme: *Si le programme est inconsistant, alors nous assumons* $\alpha \leftarrow \beta$. *Dans un autre cas, on ignore la règle.* Rappelons l'exemple suivant présenté dans [3], étant donné qu'il illustre clairement l'utilisation des CR-règles. Dans cet exemple, r_1 indique le nom de la CR-règle.

$$a \leftarrow \neg b.$$

$$\neg\, a.$$

$$r_1 : b \overset{+}{\leftarrow} .$$

Les deux premières règles sont *règles normales* et la troisième règle est une CR-règle. Ce sort de programme sera appelé CR-programmes. Nous pouvons voir que le programme sans la CR-règle est inconsistant. Pourtant, si la CR-règle s'utilise alors, la consistance est restaurée et on obtient l'answer set $\{\neg a, b\}$.

La sémantique de CR-programmes se définit en termes des *answer sets minimums généralisés*. Dans [3] les auteurs donnent une traduction de CR-programmes en programmes logiques abductifs qu'ils nomment *hard reduct*, et la sémantique se définit comme les answer sets minimums généralisés du hard reduct du programme.

[3] http://www.dbai.tuwien.ac.at/proj/dlv/K/
[4] http://www.lwb.unibe.ch/about/index.html

Les définitions suivantes se trouvent dans [3]:

Definition 2.1. [3] Un *programme logique abductif* est un couple $\langle P, A\rangle$ où P est un programme arbitraire et A un ensemble d'atomes, appelé *abducibles*.

Definition 2.2. [3] Etant donné $\langle P, A\rangle$ un programme logique abductif et $\Delta \subseteq A$. Nous disons que $M(\Delta)$ est un *answer set généralisé* de P relative A si et seulement si $M(\Delta)$ est un answer set de $P \cup \Delta$.

Definition 2.3. [3] Etant donné $M(\Delta_1)$ et $M(\Delta_2)$ deux answer sets generalisés de P relative A. Nous définissons $M(\Delta_1) < M(\Delta_2)$ si et seulement si $\Delta_1 \subset \Delta_2$ (ordre par inclusion d'ensembles).

Definition 2.4. Etant donné P un programme et étant donné A un ensemble d'atomes. $M(\Delta)$ est un *answer set minimum généralisé* de P relative à A et l'ordre $<$ si et seulement si $M(\Delta)$ est un answer set généralisé de P et elle est minime respectivement à l'ordre par inclusion d'ensembles.

Finalement, l'exemple suivant montre comment la sémantique des CR-programmes se définit en termes *answer set minimums généralisés*.

Example 2.1. Etant donné $P_1 = \{p \leftarrow \neg q.\ \ r \leftarrow \neg s.\ \ q \leftarrow t.\ \ s \leftarrow t.\ \ \leftarrow p, r.\}$ le programme de l'Exemple 4 introduit dans [3]. Nous pouvons voir que le programme P_1 n'a pas answer sets. Pourtant, pour restaurer la consistance, on peut définir un CR-programme qui ajoute quelques CR-règles. Par exemple, nous définissons le CR-programme $P_1' = P1 \cup CR_1$ où $CR_1 = \{r_1 : q \overset{+}{\leftarrow}, r_2 : s \overset{+}{\leftarrow}, r_3 : t \overset{+}{\leftarrow}\}$. Comme nous le mentionnions, la sémantique de P_1' est définie dans les termes des *answer set minimums généralisés*. Donc, en traduisant P_1' dans un *programme logique abductif* on obtient: $\langle P_1, A\rangle$ où l'ensemble de *abducibles* est $A = \{q, s, t\}$. Nous pouvons vérifier que les answer sets minimums généralisés de P_1 sont $\{q, r\}$, $\{s, p\}$ et $\{t, q, s\}$. □

2.2.3 Programmes Logiques avec Disjonction Ordonnée

Dans [7] Brewka a introduit des Programmes Logiques avec Disjonction Ordonnée, ayant pour abréviation LPODs. Pour définir ce type de programmes, le connectif $\times$, appelé *disjonction ordonnée*, s'ajoute au système des connectifs pour des formules propositionnelles présentées dans la Section 2.2.1. L'idée derrière les LPODs est d'exprimer une connaissance par défaut avec une connaissance sur les préférences d'une manière simple et élégante [9]. Comme nous l'avons mentionné dans la section 2.2.1, nous considérons dans ce travail uniquement un type de négation, la négation par échec, mais ceci n'affecte pas la validité des résultats donnée dans [7].

Formellement, un programme logique avec disjonction ordonnée est un ensemble de règles de la forme:

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k \tag{2.1}$$

où C_i, A_j y B_l sont tous des atomes ground. $C_1 \ldots C_n$ sont normalement appelés les options d'une règle et sa lecture intuitive est la suivante:

Si le corps est vrai et C_1 est possible, alors C_1; si C_1 n'est pas possible, alors C_2; $\ldots$; si aucun de $C_1, \ldots, C_{n-1}$ n'est possible alors C_n. On peut préciser quelques cas spéciaux, $n = 0$ (restrictions), $n = 1$ (programmes étendus), $m = k = 0$ (faits). La sémantique des programmes logiques avec disjonction ordonnée sera présentée dans la Section 2.3.1 où une extension des programmes logiques avec disjonction ordonnée est présentée. Actuellement, il y a une modification de SMODELS[5] qu'on peut utiliser pour le calcul des answer sets préférés des programmes logiques avec disjonction ordonnée, appelée PSMODELS[6] [9].

[5] http://www.tcs.hut.fi/Software/smodels/
[6] http://www.tcs.hut.fi/Software/smodels/priority/

2.2.4 Planification dans Answer Sets

Dans un problème de planification, nous nous intéressons à la recherche d'une séquence d'actions qui conduise d'un état initial donné à un état qui définit le but. Pour spécifier totalement un problème de planification, nous avons besoin d'indiquer que des actions se permettent dans un plan. La planification dans Answer Sets utilise des langages d'actions pour spécifier les effets des actions et ils sont utilisés en particulier pour modeler des problèmes de planification [16]. Certains de ces langages d'action sont $\mathcal{A}, \mathcal{B}$, et $\mathcal{C}$ [16]. D'autre part, un problème de planification spécifique à ce type de langage a une codification naturelle et facile dans Answer Set Programing.

Langage $\mathcal{A}$. L'alphabet du langage $\mathcal{A}$ consiste en deux ensembles non vides et de disjonctions de symboles **F** appelé l'ensemble des fluents, et **A** appelé l'ensemble des actions. Intuitivement, un fluent exprime une caractéristique d'un objet dans un monde et fait partie de la description des états du monde. Une *fluent littéral* est un fluent ou un fluent précédé de $\sim$. Un *état* σ est un ensemble de fluents.

Disons qu'un fluent f se réalise dans un état σ si $f \in \sigma$. Disons qu'un fluent littéral $\sim f$ se réalise dans σ si $f \notin \sigma$. Quand les actions sont exécutées avec succès, elles changent l'état du monde. Les situations sont des représentations de l'histoire de l'exécution des actions. La situation initiale $[a_n, \ldots, a_1]$ correspond à l'histoire où l'action a_1 s'exécute dans la situation initiale, suivie par a_2, et jusqu'à a_n. Il y a une relation simple entre des situations et des états. Dans chaque situation quelques fluents sont vrais et d'autres sont faux.

Le langage $\mathcal{A}$ peut se diviser en trois sous-langages *Langage pour la description du domaine*, *Langage d'observation*, et *Langage de consultation*. Si on a un problème de planification, il peut être exprimé dans un langage d'actions au moyen d'un triple (D, O, G) tel que D est la description du domaine, O est l'ensemble des observations

et G est l'ensemble des conditions du but. D'autre part, il est possible de définir une codification dans Answer Sets de ce problème de planification spécifié dans un langage d'actions, indiqué comme $\pi(D, O, G, l)$, où l est un numéro entier positif qui correspond à la longitude du plan, c'est à dire le numéro des actions à réaliser pour atteindre le but.

Il est donc possible d'obtenir la solution du problème de la planification (les plans) i.e. des answer sets de $\pi(D, O, G, l)$. L'idée de base est que nous décidons de la longitude du plan pour énumérer les circonstances des actions jusqu'aux points du temps qui correspondent à cette longitude et codifier les buts comme restrictions par rapport au temps $l + 1$, tel que les answer sets qui codifient les ocurrences des actions qui ne satisfont pas le but dans le temps sont éliminés. Ainsi chaque answer sets qui survit aux restrictions correspond à un plan. Il est important de mentionner que dans la codification dans Answer Sets des problèmes de la planification exprimés en langage $\mathcal{A}$, si f est un fluent littéral négatif $\sim g$ donc f sera codifié comme $neg(g)$, où $neg(g)$ correspond à la négation classique.

Les détails de ces sous-langages et la manière d'obtenir la codification d'un problème de planification d'une spécification du langage d'actions se trouvent dans [16, 4]. Nous présentons seulement un exemple qui démontre comment spécifier un problème simple de la planification du monde des blocs dans un langage $\mathcal{A}$ provenant de [4].

Example 2.2. Le problème simple de la planification du monde des forêts.

1. Fluentes: $on(X, Y), ontable(X), clear(X), holding(X), handempty$
2. Actions: $pick\_up(X), put\_down(X), stack(X, Y), unstack(X, Y)$
3. Conditions d'exécution, dénotées par D:
   executable $pick\_up(X)$ **if** $clear(X), ontable(X), handempty.$
   executable $put\_down(X)$ **if** $holding(X)$.

executable $stack(X, Y)$ **if** $holding(X), clear(Y)$.

executable $unstack(X, Y)$ **if** $clear(X), on(X, Y), handempty$.

4. Propositions d'effet.

   $pick\_up(X)$ **causes** $\sim ontable(X), \sim clear(X), holding(X), \sim handempty$.

   $put\_down(X)$ **causes** $ontable(X), clear(X), \sim holding(X), handempty$.

   $stack(X, Y)$ **causes** $\sim holding(X), \sim clear(Y), clear(X), handempty, on(X, Y)$.

   $unstack(X, Y)$ **causes** $holding(X), clear(Y), \sim clear(X), \sim handempty, \sim on(X, Y)$.

5. Considérant *block* comme $\{a, b, c, d\}$ les conditions initiales, dénotées par I, sont décrites comme:

   $initially(clear(a))$. $initially(clear(c))$. $initially(clear(d))$. $initially(handempty)$.

   $initially(ontable(c))$. $initially(ontable(b))$. $initially(on(a, b))$. $initially(on(d, c))$.

6. Les conditions de but, dénotées par G, sont l'ensemble de de littéraux

   $\{on(a, c), on(c, b), ontable(d)\}$. □

Le Langage $\mathcal{B}$ apparaît quand le langage $\mathcal{A}$ est étendu pour permettre *des propositions de cause statiques*. En utilisant des propositions de causes statiques, on peut exprimer une connaissance au sujet des états qui sont possibles ou valides. Cette connaissance peut être utilisé indirectement pour déduire des effets d'actions, le fonctionnement des actions ou les deux. On peut trouver plus de détails sur les propositions de cause statiques et la codification de problèmes exprimés en langage $\mathcal{B}$ dans [16, 4].

2.2.5 Un langage pour des préférences de plans: le Langage $\mathcal{PP}$

Le langage $\mathcal{PP}$ [43] est un langage pour lé spécification de préférences del utilisateur avec une implémentation dans la programmation logique, basée sur Answer Set Programming. Le langage permet de spécifier des préférences entre des plans possibles et

également des préférences temporelles sur les plans. Les préférences qui représentent le temps sont exprimés à l'aide des connectifs temporels *next, always, until* et *eventually.* Il y a trois classes de préférences dans $\mathcal{PP}$ [43] *desirs basiques*, *préférences atomiques* et *préférences générales* . Dans ce travail, nous étudions les deux premières classes car elles ont été utiles pour exprimer des préférences dans les plans d'évacuation. Un *désir basique*, indiqué comme φ, est une formule de $\mathcal{PP}$ qui exprime une préférence sur une trajectoire relative à l'exécution d'une certaine action ou relative aux états de la trajectoire quand l'action se réalise.

Une *préférence atomique* représente un ordre entre formules de désir basique. Il indique que les trajectoires qui satisfont φ_1 sont préférables à celles qui satisfont φ_2, etc. En clair, les formules de désir basique sont des cas spéciaux de préférences atomiques.

2.3 Contribución

Rappelons que l'objectif de notre travail est de faire des recherches et d'évaluer les capacités de Answer Set Programming pour représenter des situations de désastre ayant pour but d'aider à la définition de plans d'évacuation. Pour cette raison, nous commençons à analyser comment l'information géographique de la zone de désastre à modeler peut être traduite dans un format que Answer Sets est capable de comprendre. Nous continuons en étudiant et en appliquant diverses approches de Answer Sets qui sont utiles pour définir les plans d'évacuation tels que CR-Programmes, les programmes logiques avec disjonction ordonnée, la planification dans Answer Sets, le langage pour des préférences de plans et les answer sets minimums généralisés. De manière spécifique, nous proposons l'extension des Programmes Logiques avec Disjonction Ordonnée et nous voyons son utilité. Nous allons présenter comment et pourquoi les approches mentionnées de Answer Sets s'appliquent pour définir des plans d'évacuation et nous

présentons aussi quelques disaventajes des approches traitées dans ce travail.

2.3.1 Programmes Logiques avec Disjonction Ordonnée Etendue

Dans [7] la tête des règles avec disjonction ordonnée étendue est définie en termes de ground littérales. Dans cette section, la tête et le corps des règles avec disjonction ordonnées étendues sont définis en termes de formules propositionnelles bien formées. Nous considérons qu'une syntaxe plus simple pour ces règles pourrait nous donner quelques avantages. Par exemple, l'utilisation d'expressions hiérarchisées pourrait simplifier la tâche d'écrire des programmes logiques et d'améliorer sa lisibilité puisqu'il nous permettrait d'écrire des règles plus courtes et plus naturelles.

Definition 2.5. Une *règle avec disjonction ordonnée étendue* est une formule propositionnelle bien formée comme on la définie dans la Section 2.2.1, ou une formule de la forme:

$$f_1 \times \ldots \times f_n \leftarrow g \tag{2.2}$$

où $f_1, \ldots, f_n, g$ sont des formules propositionnelles bien formées. Un *programme avec disjonction ordonnée étendue* est un ensemble fini de règles avec disjonction ordonnée étendue. □

Les formules $f_1 \ldots f_n$ sont généralement appelées les options d'une règle et leur lecture individuelle est la suivante: si le corps est vrai et f_1 est possible, alors f_1; si f_1 n'est pas possible, alors f_2; ...; si aucun de $f_1, \ldots, f_{n-1}$ n'est possible, alors f_n. Le cas particulier où toute f_i est un littéral et g est un ensemble de littérales correspondant aux programmes avec disjonction ordonnée originales comme a été présenté pour Brewka dans [7], et nous l'appellerons *programme avec disjonction ordonnée standard*. Rappelons que les programmes avec disjonction ordonnée standard de Brewka utilisent le connectif pour la négation forte.

Nous considérons ici seulement un type de négation (négation par échec) mais ceci n'affecte pas les résultats donnés dans [7]. Si $n = 0$ la règle est en plus une restriction, c'est à dire, $\bot \leftarrow g$. Si $n = 1$ est une règle étendue et si $g = \top$ la règle est un fait et peut s'écrire comme $f_1 \times \ldots \times f_n$. Nous présentons maintenant la sémantique des programmes avec disjonction ordonnée étendue. La majorité des définitions présentées ici on été définies dans [7, 9].

Definition 2.6. [7] Etant donné $r := f_1 \times \ldots \times f_n \leftarrow g$ une règle avec disjonction ordonnée étendue. Pour $1 \leq k \leq n$ la *k-th option de r* est définie comme suit: $r^k := f_k \leftarrow g, \neg f_1, \ldots \neg f_{k-1}$ □

Definition 2.7. [7] Etant donné P un programme avec disjonction ordonnée étendue. P' est un *programme partagé de* P s'il est obtenu par le remplacement de chaque règle $r := f_1 \times \ldots \times f_n \leftarrow g$ dans P pour une de ses options $r^1, \ldots, r^k$. M est un answer set de P si et seulement si est un answer set[7] d'un programme partagé P' de P. □

Definition 2.8. [7] Etant donné M un answer set d'un programme avec disjonction ordonnée étendue P et étant donné $r := f_1 \times \ldots \times f_n \leftarrow g$ une règle de P. Nous définissons le *degré de satisfaction de r relative à M*, indiqué par $deg_M(r)$, comme suit:
—si $M \cup \neg(\mathcal{L}_P \setminus M) \not\vdash_{\mathrm{I}} g$, alors $deg_M(r) = 1$.
— si $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} g$ alors $deg_M(r) = min_{1 \leq i \leq n} \{i \mid M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} f_i\}$. □

Le théorème suivant indique la relation entre les answer sets d'un programme avec disjonction ordonnée étendue et le degré de satisfaction de chaque règle dans le programme.

Theorem 2.2. *[7] Etant donné P un programme avec disjonction ordonnée étendue. Si M est un answer set de P alors M satisfait toutes les règles dans P à un certain*

[7] notons que du fait que nous considérons une négation forte, il est impossible d'avoir des answer sets inconsistants.

degré.

Definition 2.9. [9] Etant donné P un programme avec disjonction ordonnée étendue et M un ensemble de littérals. Nous définissons $S^i_M(P) = \{r \in P \mid deg_M(r) = i\}$.

Grâce aux définitions présentées, nous allons introduire deux types différents de relations de préférence entre les answer sets d'un programme avec disjonction ordonnée étendue: la préférence basée sur l'inclusion d'ensembles et la préférence basée sur la cardinalité des ensembles.

Definition 2.10. [9] Etant donné M et N answer sets d'un programme avec disjonction ordonnée étendue P. Alors nous disons que M est *préféré par inclusion* à N, indiqué comme $M >_i N$, si et seulement si il y a un i tel que $S^i_N(P) \subset S^i_M(P)$ et pour tout $j < i$, $S^j_M(P) = S^j_N(P)$. Nous disons que M est *préféré par cardinalité* à N, indiqué comme $M >_c N$, si et seulement si il y a un i tel que $|S^i_M(P)| > |S^i_N(P)|$ et pour tout $j < i$, $\left|S^j_M(P)\right| = \left|S^j_N(P)\right|$. □

Definition 2.11. [9] Etant donné M un answer set d'un programme avec disjonction ordonnée étendue P. M est un *answer set préféré par inclusion* de P s'il n'y a aucun answer set M' de P, $M \neq M'$, tel que $M' >_i M$. M est un *answer set préféré par cardinalité* de P s'il n'y a pas M' answer set de P, $M \neq M'$, tel que $M' >_c M$. □

2.3.2 Answer Set Minimum Généralisé et Disjonction ordonnée

Nous proposons maintenant une caractérisation des answer sets minimums généralisés d'un programme en terme de programme avec disjonction ordonnée standard. Ce résultat théorique prouve que les deux programmes abductifs et les programmes avec CR-règles peuvent être représentés correctement en utilisant la disjonction ordonnée.

Definition 2.12. Etant donné $< P, A >$ un programme logique abductif. Alors, nous définissons une traduction à un programme avec disjonction ordonnée standard, indiqué par $ord(P, A)$, comme suit : D'abord pour un littéral $a \in A$ la règle r_a est définie comme $a^\bullet \times a$, où $a^\bullet$ est un littéral qui n'apparaît pas dans le programme original. Nous définissons $ord(P, A) = P \cup \{r_a \mid a \in A\}$. □

Lemma 2.1. *$M \cap \mathcal{L}_P$ est un answer set généralisé du programme logique abductif $< P, A >$ si et seulement si M est un answer set de $ord(P, A)$.* □

Le théorème suivant prouve la validité de notre traduction pour la sémantique de programmes logiques abductifs.

Theorem 2.3. *Etant donné $< P, A >$ un programme logique abductif et M un ensemble d'atomes. $M \cap \mathcal{L}_P$ est un answer set minime ou généralisé de $< P, A >$ si et seulement si M est un answer set préféré par inclusion de $ord(P, A)$.* □

2.3.3 Préférences en termes de programme avec Disjonction Ordonnée Étendue

Nous pouvons spécifier un ordre entre les answer sets d'un programme avec disjonction ordonnée étendue relative à une liste d'atomes.

Definition 2.13. Etant donné P un programme normal. Etant donné M et N deux answer sets de P. Etant donné $C = [c_1, c_2, \ldots, c_n]$ un ensemble ordonné de formules. L'answer set M *est préféré à l'answer set* N *respectivement à* $C \cup P$ (indiqué comme $M <_{C \cup P} N$) s'il existe $i = min(1 \leq k \leq n)$ tel que $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} c_i$ et $N \cup \neg(\mathcal{L}_P \setminus N) \nvdash_{\mathrm{I}} c_i$; et pour tout $j < i$, $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} c_j$ et $N \cup \neg(\mathcal{L}_P \setminus N) \vdash_{\mathrm{I}} c_j$ ou $M \cup \neg(\mathcal{L}_P \setminus M) \nvdash_{\mathrm{I}} c_j$ et $N \cup \neg(\mathcal{L}_P \setminus N) \nvdash_{\mathrm{I}} c_j$. □

Proposition 2.1. *Etant donné $C = [c_1, c_2, \ldots, c_n]$ un ensemble ordonné de formules; alors $<_{C \cup P}$ est un ordre partiel.* □

Un answer set M d'un programme normal P *est préférable* s'il n'y a pas d'autre answer set N de P qui soit préféré à M.

Example 2.3. Etant donné $C = [b, c]$ un ensemble ordonné d'atomes. Etant donné P le programme suivant normal: $\{a \leftarrow .\ \ c \leftarrow \neg b.\ \ b \leftarrow \neg c.\}$. Nous pouvons vérifier que P a deux answer sets, $\{a, b\}$ et $\{a, c\}$. Et également que l' answer set $\{a, b\}$ *est préféré à l'answer set* $\{a, c\}$ *relative à* C, i.e., $\{a, b\} <_{C \cup P} \{a, c\}$ et que $\{a, b\}$ est aussi *l' answer set préféré.* □

Definition 2.14. Etant donné P un programme normal et $C = [c_1, c_2, \ldots, c_n]$ un ensemble ordonné d'atomes tel que $C \subseteq \mathcal{L}_P$. Nous définissons une *règle avec disjonction ordonnée étendue définie par* C, indiqué comme r_C, comme suit: $r_C := \neg\neg c_1 \times \neg\neg c_2 \times \ldots \times \neg\neg c_n \times all\_pref$ tel que $all\_pref \notin \mathcal{L}_P$. □

Lemma 2.2. *Etant donné P un programme normal et $C = [c_1, c_2, \ldots, c_n]$ un ensemble ordonné d'atomes tel que $C \subseteq \mathcal{L}_P$. Etant donné r_C la règle avec disjonction ordonnée étendue définie par C. Alors M est un answer set préféré par inclusion de $P \cup r_C$ si et seulement si $(M \cap \mathcal{L}_P)$ est l'answer set préféré respectivement à $C \cup P$.* □

Example 2.4. Nous considérons encore une fois le programme P de l'Exemple 2.3 et l'ensemble ordonné d'atomes $C = [b, c]$. Alors, $P \cup r_C$ est le programme suivant $\{a \leftarrow .\ \ c \leftarrow \neg b.\ \ b \leftarrow \neg c.\ \ \neg\neg b \times \neg\neg c \times all\_pref.\}$. Nous pouvons vérifier que $\{a, b\}$ est l'answer set préféré respectivement à $C \cup P$ et que c'est aussi un answer set préféré par inclusion de $P \cup r_C$. □

Il vaut la peine de mentionner que aucun de deux: lxecution de *PSMODELS*[8] [9] ou l'utilisation de la definition donée par Brewka (disjonction ordonnée) permettent

[8] http://www.tcs.hut.fi/Software/smodels/priority/

d'obtenir les answer sets préférés par inclusion d'ensembles pour des programmes avec disjonction ordonnée étendue. La raison en est que la définition donnée par Brewka pour la disjonction ordonnée a des restrictions syntaxiques.

Pourtant, en particulier quand le programme a des règles de disjonction ordonnée étendue qui utilisent des littérales niés, nous pouvons facilement traduire ce programme à un programme avec disjonction ordonnée étendue standard et donc utiliser *PSMOD-ELS* pour obtenir les answer sets préférés [46]. Dans la définition et le lemme suivants, les atomes $a^\bullet$, a°, sont des atomes qui n'apparaissent pas dans le programme original P.

Definition 2.15. Etant donné $\neg\neg a$ un littéral nié. Nous définissons l'ensemble des règles avec disjonction ordonnée associé à $\neg\neg a$ comme suit:

$R(\neg\neg a) := \{ \leftarrow \neg a, a^\bullet. \quad a^\bullet \leftarrow \neg a^\circ. \quad a^\circ \leftarrow \neg a. \quad \leftarrow a, a^\circ. \ \}.$ □

Lemma 2.3. *Etant donné P un programme normal et étant donné $C = [c_1, c_2, \ldots, c_n]$ un ensemble ordonné d'atomes tel que $C \subseteq \mathcal{L}_P$. Etant donné $C^\bullet = [c_1^\bullet, c_2^\bullet, \ldots, c_n^\bullet]$ un ensemble ordonné d'atomes tel que $\{c_i^\bullet \in C \mid 1 \leq i \leq n\} \cap \mathcal{L}_P = \emptyset$. Etant donné r_C la règle avec disjonction ordonnée étendue définie de $C^\bullet$. Etant donné $A = \bigcup_{c_i \in C \text{ and } 1 \leq i \leq n} R(\neg\neg c_i)$. Alors M est un answer set préféré par inclusion de $P \cup r_C^\bullet \cup A$ si et seulement si $M \cap \mathcal{L}_P$ est un answer set préféré par inclusion de $P \cup r_C$.* □

2.3.4 Information géographique

La majorité des informations nécessaires pour modeler le problème de l'évacuation doit s'obtenir des données géographiques. Nous proposons un procédure pour construire la connaissance de la région de danger à partir de l'information géographique. Ce procédé est le résultat de notre propre expérience de travail avec ce type de données. Il vaut la

peine de mentionner que nous prouvons dans ce travail tous nos résultats avec une petite partie des données géographiques réels de la zone de danger du volcan Popocatépetl, puisque les données réelles totales de cette zone ont été obtenues pratiquement à la fin de ce travail. Le procédé peut être décrit dans les termes suivant: Extraction de l'information descriptive [1] qui fait partie de l'information géographique, à l'aide d'outils qui traitent ce type de données, réparation d'inconsistances, et finalement ajout de l'information descriptive additionnelle.

2.3.5 Problème de plan d'évacuation

Pour obtenir les plans de l'évacuation, nous commençons par modeler le problème des plans d'évacuation en utilisant Answer Set Planning.

Normalement, dans une zone de risque, les autorités définissent des voies d'évacuation. Chaque voie d'évacuation commence dans la zone à risque, traverse d'autres lieux où le risque est moins important et arrive finalement dans une zone hors de danger. Parfois, le lieu hors de danger correspond à un refuge. Chaque refuge contient des provisions suffisantes pour la population. Pourtant, certains dangers qui peuvent accompagner un désastre résultent du blocage d'une partie des voies d'évacuation prédéfinies. Il est donc nécessaire de définir des plans alternatifs d'évacuation. Le *problème pour définir des plans d'évacuation alternatifs* (AEP-problème) peut être énoncé comme suit:

> *Il existe un ensemble de voies d'évacuation pour les gens qui vivent dans la zone à risque. Chaque voie d'évacuation prédéfinie peut avoir différents points de départ mais un seul point d'arrivée. Dans le cas où une partie d'une voie d'évacuation prédéfinie est inaccessible, les évacués doivent alors chercher une trajectoire alternative. Cette trajectoire alternative peut appartenir ou non à une autre voie d'évacuation. Si elle n'appartient pas à une voie d'évacuation, il est alors préférable, dans un ordre décroissant, de rejoindre un endroit faisant partie d'une voie d'évacuation ou un refuge ou un lieu hors de la zone de danger.*

Nous présentons deux solutions au problème de la définition de plans d'évacuation alternatifs, les deux utilisent Answer Set Planning et certaines approches dans Answer Sets pour les préférences. Appliquer des préférences se révèle une manière simple et efficace d'obtenir une solution appropriée à un problème donné. Par exemple, dans le cas où nous avons un ensemble de désirs pour un plan d'évacuation que nous voudrions satisfaire, mais qu'en même temps, ils ne peuvent pas être satisfaits tous simultanément, on pourrait appliquer des préférences. Un autre exemple est quand, face à un problème donné de planification, on pourrait obtenir un nombre élevé de solutions. Pour cela, il est nécessaire d'exprimer des préférences pour choisir l'un des plans possibles d'évacuation. Les solutions présentées dans ce travail veillent à la spécification de telles préférences parmi les plans éventuels. Nous avons étudié deux approches dans answer sets: *CR-régles* et le Langage $\mathcal{PP}$.

En utilisant les CR-règles

Nous proposons d'élargir un problème de planification modelée en utilisant Answer Set Planning et en lui ajoutant les CR-règles [47, 50]. L'idée est d'obtenir les plans d'évacuation en profitant de la définition d'une CR-règle et d'utiliser ces règles seulement s'il n'y a pas moyen d'obtenir un plan d'évacuation quand une partie de la voie prédéfinie est bloquée. Pour cela, les CR-règles sont définies sur les actions utilisées pour modeler un problème. Dans le cas où une CR-règle s'applique, la consistance est restaurée, ce qui permet d'obtenir des plans d'évacuation alternatifs. En particulier, la CR-règle que nous proposons d'ajouter est la suivante:

$$r_2 : action(travel(B, P, Q, R')) \stackrel{+}{\leftarrow} bus(B), road(P, Q, R').$$

L'idée de la CR-règle r_2 est qu'il est possible de voyager de P à Q s'il y a un segment de chemin de P à Q. En contraste avec l'action normale *travel* définie dans $\pi(D, O, G, l)$,

cette CR-règle ne vérifie pas si le chemin de P à Q fait partie d'une voie d'évacuation, c'est à dire, cela n'est pas important si R' est égal a zéro ou pas. De cette manière, r_2 nous permet de trouver les plans d'évacuation alternatifs. Voir les Exemples B.1, B.2, et B.3 dans l'Appendice B qui montrent comment nous pouvons utiliser les CR-règles pour obtenir des plans d'évacuation alternatifs. Nous soulignons que dans un programme donné avec CR-règles qui obtient des plans d'évacuation alternatifs, nous pouvons traduire le CR-règles en accord avec la Définition 2.12 pour obtenir un programme avec disjonction ordonnée standard et donc utiliser PSMODELS[9] . Pour obtenir les answer sets préférés qui correspondent aux plans d'évacuation alternatifs.

En utilisant le Langage $\mathcal{PP}$

Comme nous l'avons vu, l'utilisation des CR-règles permet d'obtenir des plans d'évacuation alternatifs, pourtant, ces plans alternatifs ne considèrent aucune autre caractéristique de la trajectoire qu'ils suivent. Pour cela, nous considérons le langage $\mathcal{PP}$ pour exprimer des préférences de plans alternatifs à divers niveaux. Nous décidons d'utiliser $\mathcal{PP}$ parce qu'il nous permet d'exprimer des préférences pour les plans où la satisfaction de ces préférences dépend du temps. Nous considérons que particulièrement dans la définition des plans d'évacuation, il est très utile d'exprimer des préférences en termes de temps. Par exemple, *toujours* on a préféré évacuer les gens en danger en suivant les voies d'évacuation prédéfinies. Pourtant, si *éventuellement* une partie des voies d'évacuation est bloquée, alors les évacués se déplaceraient hors de la zone de danger jusqu'à arriver à un refuge. Nous démontrons aussi que les règles disjonctives étendues avec des littérales niés pourraient être utiles pour permettre une codification plus simple afin d'obtenir les plans préférés relatives à une préférence exprimée dans langage $\mathcal{PP}$. L'Exemple B.4 dans l'Appendice B montre l'usage du langage $\mathcal{PP}$ pour exprimer les

[9] http://www.tcs.hut.fi/Software/smodels/priority/

préférences parmi les plans d'évacuation.

2.3.6 En étendant le Langage $\mathcal{PP}$

Lors de l'utilisation de $\mathcal{PP}$ pour exprimer des préférences, nous avons réalisé que certaines préférences ne peuvent s'exprimer d'une manière simple et naturelle puisque ces préférences s'avèrent être très grandes. Pour cette raison, pour avoir une représentation plus naturelle de cette classe de préférences, nous définissons que le langage $\mathcal{PP}^{par}$ $\mathcal{PP}^{par}$ est une extension du langage $\mathcal{PP}$ dont les connectifs proportionnels et temporels permettent de représenter de manière compacte les préférences qui ont une caractéristique commune en particulier. Voir l'Exemple B.5 de l'Appendice B.

Nous avons également pris en considération le fait que le langage $\mathcal{PP}$ pourrait être avantagé pour tout le déroulement théorique de la Logique Propositionnelle Linéale Temporelle (LTL) pour exprimer des préférences. Pour cela, nous avons analysé comment on pourrait établir la relation entre le langage $\mathcal{PP}$ et LTL.

2.3.7 Contenu Sémantique

Nous avons présenté la notion de *Contenu Sémantique* d'un programme comme point de vue alternatif pour obtenir différentes sémantiques dans answer sets d'un programme. Nous montrons en particulier comment obtenir les *answer sets standard*, les *answer sets généralisés*, les *answer sets minimums généralisés* et une nouvelle sémantique que nous nommons *answer sets partiels*. Finalement, nous proposons d'utiliser les answer sets partiels pour obtenir des plans partiels d'évacuation.

2.4 Conclusions

Dans cette thèse, nous avons proposé d'étudier Answer Set Programming et d'évaluer ses capacités à représenter des situations de désastre dans le but d'aider à définir des plans d'évacuation. Nous sommes arrivés à la conclusion qu'en utilisant Answer Set Programming il est possible de représenter des situations de désastre et surtout de profiter de ses capacités, des diverses approches qui existent maintenant pour obtenir des plans d'évacuation. Pour finir, nous allons présenter quelques desaventages des approches dont nous essayons d'améliorer dans ce travail.

Un fait intéressant pour la recherche dans le futur est l'utilisation de certaines approches Answer Sets destinées à devenir la génération la plus efficace des answer sets dans la génération des plans d'évacuation. Par exemple, relativement aux conditions de la mémoire, on pourrait utiliser des mécanismes plus efficaces comme il est proposé dans [5]. Une autre direction que pourraient prendre les recherches serait de modeler des domaines dynamiques en utilisant Answer Sets pour générer les plans d'évacuation où les caractéristiques éminentes des évènements exogènes (par exemple une explosion ou la coulée de la lave) qui apparaissent dans la zone de danger et l'information sur cette zone seraient considérées. Les approches dans Answer Sets utilisées dans [3] pour modeler des systèmes dynamiques et dans [10] pour modeler des coulées de lave pourraient nous aider dans ce sens. Enfin, il serait intéressant d'élargir notre utilisation du Contenu Sémantique pour trouver d'autres variantes de la sémantique de Answer Sets, tel que les programmes avec disjonction ordonnée et alors, analyser leur utilisation pour obtenir les plans d'évacuation. Par exemple, dans [48] quelques cas sont illustrés en utilisant les Anwer Sets partiels définis en termes de leur contenu sémantique.

Introduction

In 1985, the United Nations Disaster Relief Coordinator (UNDRO) and United Nations Educational Scientific and Cultural Organization (UNESCO) remarked that throughout the world has been growing steadily the awareness of being prepared for disaster situations and to provide protection against them, rather than simply to await and endure them [30].

Currently, Geographical Information Systems (GISs)—computer systems that manage, display, and support analysis of geographical reference data— are increasingly being used to fill many crisis management needs [23, 42]. All phases of crisis management deal with many location-specific details, drawn from sources including remote sensing and Global Positioning System (GPS) data on the region of an event and its effects. A GIS assists in managing such information by associating related geographic information and integrating multiple geographic information elements during a crisis. In particular, a GIS can update its database during the course of a disaster to reflect what is known about the environment and situation during the response efforts; the spatial data of the GIS can be reproduced electronically for distribution and access; and a GIS provides ways to combine many types of data of value for crisis management [23].

However, once the data about a disaster situation are updated, transmitted, displayed or combined by a GIS, people involved in protection against disaster situations

must make the decisions about preparing and executing evacuation plans for potential causes of disaster. Hence, it would be desirable to develop an "intelligent GIS" capable of obtaining and analyzing evacuation plans based on knowledge about their particular environment, the geographic data and their own capabilities, and of exchanging information and services with similar systems as well as with humans. Specifically, an "intelligent GIS" should has the following capabilities:

- *Consider geographic data.* The geographic data used in a crisis varies according to location. Some states have extensive GISs of their own with up-to-date details. These systems include information on evacuation routes and location of emergency shelters and estimates of populations-at-risk at various times of the day. Other considered sources include background GIS maps, including roads and locations of industry; census data and some commercially available data sets; and aerial photographs for the area.
- *Model evacuation plan problems.* In order to model evacuation plan problems, the system should be able to represent different initial scenarios, the possible actions with their executability conditions and effects, the different evacuation plan goals, and restrictions of time on all possible plans.
- *Consider dynamic environments.* Based on the kind of disaster, it may happen that the topography itself changes or the built environment, including, roads, buildings, and utility services, is affected. As we mentioned, a GIS is able to update its database during the course of a disaster to reflect what is known about the environment. Additionally, if the current evacuation plan cannot be performed due to the changes such as damaged roads, then the plan also has to be updated.
- *Expressing exceptions.* It is necessary to express all possible exceptions when a plan is prepared or executed, for instance we could express that "Normally, a car

travels by predefined evacuation routes but exceptionally, if some segment of roads are blocked, it may travel by other roads".

- *Incomplete knowledge.* It is normal that humans and machines get conclusions under incomplete knowledge. In a similar way, it is necessary to define evacuation plans under incomplete knowledge. For instance, a car can travel by a segment of road if we do not have the explicit information that it is blocked.
- *Expressing preferences.* Applying preferences result to be a very natural and effective way to obtain an appropriate solution for some problems. For instance, in situations where we have a set of desiderata about an evacuation plan that we would like to satisfy and not all of them can be simultaneously satisfied. Another example is when given a planning problem, we may obtain a high number of solutions. Then it is necessary to express preferences to choose some of the possible evacuation plans.
- *Simple to specify planning problems.* People that make the decisions about preparing and executing evacuation plans do not have to worry about the way to deal with the system that should help them define the evacuation plan. Then the system should allow define evacuation plans using a simple English-like syntax to specify the planning problems.
- *Efficiency.* If the system is used to define evacuation plans in real time, then the system should be efficient to generate the plans. If the system is used in pre-defining planning, then it is not necessary to have a very efficient system.
- *Express restrictions.* All evacuation plans have different kinds of restrictions depending on the region and service facilities. For instance, if a segment of road have a capacity of a number of vehicles by unit of time then the system should be able to consider this kind of restrictions in order to define the evacuation plans.

- *Consider generality.* If some changes in the geographic data, preferences, restrictions occur then, the way in that the system obtains the evacuation plan does not have to change.

We think that a good formalism to explore to develop such "intelligent GISs" is a logical approach. Currently, one of these formalisms is a declarative knowledge representation and logic programming language called Answer Set Programming [14]. The original definition of answer sets was given by Gelfond and Lifschitz in 1988 [14]. Answer Set Programming is the realization of much theoretical work on Non-monotonic Reasoning and AI applications. It represents a new paradigm for logic programming that allows, using the concept of negation as failure, to handle problems with default knowledge and produce non-monotonic reasoning. Two popular software implementations to compute answer sets are DLV[10] and SMODELS[11] . The efficiency of such programs allowed to increase the list of practical applications in the areas of planning, logical agents and artificial intelligence.

Hence, the motivation of our work is to contribute towards the development of such "intelligent GISs". Specifically, the objective of our work is to *investigate and evaluate the capabilities of Answer Set Programming (ASP) to represent disaster situations in order to give support in defining evacuation plans.*

Moreover, the motivation of our work is based on the idea that Answer Set Programming presents most of the capabilities that we mentioned above about an "intelligent GIS" should have. Specifically,

- It is possible that Answer Set Programming considers geographic information. Of course, geographic information should be translated into a format that answer sets can understand. In particular in this work we propose a procedure to translate

[10] http://www.dbai.tuwien.ac.at/proj/dlv/
[11] http://www.tcs.hut.fi/Software/smodels/

geographic information and to construct the hazard zone background knowledge from this information.

- Currently there exists *Answer Set Planning* that provides a natural and elegant way to model planning problems [16]. Answer Set Planning allows us to represent states, initial states, actions, executability conditions and effects of actions, and goals in order to obtain the feasible plans. The key element of answer set planning is the representation of a dynamic domain in the form of a "history program"— a program whose answer sets represent possible "histories", or evolutions of the system, over a fixed time interval.
- Answer Set Programming and Answer Set Planning use the concept of negation as failure that allows us to express exceptions and represent incomplete knowledge.
- Currently there exist different approaches to express preferences in answer sets. In particular in this work we studied and used two of them.
- Answer Set Planning is based on different action languages that are formal models of parts of the natural language used for describing the effects of actions [16, 4]. Then, specifying a planning problem could result easy and natural for people involved in defining evacuation plans.
- Answer Set Programming allows to express restrictions. In particular Answer Set Planning allows express restrictions about the executability of actions.
- Finally, given a planning problem specified in an action language, it is possible to define a general way to obtain its respective answer set encoding. In Answer Set Planning is proposed to divided the encoding in two parts: the *domain dependent part*, and the *domain independent part*. The former is a direct translation of the domain description in the action language and the latter is independent of the domain and may be used for planning with other domains. In this way, depending

on the problem we only have to define the domain dependent part and add the domain independent part in order to obtain the feasible plans.

In the contribution of this work we present different practical and theoretical results related to the capabilities of Answer Set Programming (ASP) to represent disaster situations in order to give support in defining evacuation plans.

This thesis is divided into two parts respectively describing its background and its contributions.

The Background have the following chapters:

Chapter 3 presents a synthesis of information related to Volcanic emergency management recommended by the UNESCO and in particular the volcano Popocatépetl case.

Chapter 4 presents a synthesis of the theoretical results needed to understand the contribution of our work. It presents the Answer set programming framework, CR-Programs and Abductive logic programs, and Ordered Disjunction programs.

Chapter 5 starts presenting an overview about one of the AI planning approaches to model and solve planning problems in a natural and elegant way is *Answer Set Planning* [16]. It also presents an abstract from [19] about the analysis of a general *transportation* problem that generalizes planning in several classical domains since, some of the evacuation planning problems can be similar cases of this family of problems. Finally, this chapter presents a brief overview of Language $\mathcal{PP}$ for planning preferences.

The Contribution starts with a discussion about some lacks in existing work that this work want to address. Then this part continues by presenting the following chapters:

Chapter 6 introduces extended ordered disjunction programs such that ordered disjunctions is extended to wider classes of logic programs. A characterization of minimal generalized answer sets in terms of ordered disjunction programs is also proposed. This

theoretical result proves that both abductive programs and programs with CR-rules can be properly represented using ordered disjunction. This chapter also proposes to specify and compute a preference ordering among the answer sets of a program with respect to an ordered list of atoms using a extended ordered disjunction programs with double negation [46]. In this chapter is also proposed a second application of extended ordered disjunction programs with double default negation to obtain the maximal answer set of a program. Finally, we show how to compute the preferred answer sets for extended ordered programs using PSMODELS.

Chapter 7 analyzes the information which would be ideal to have to model an evacuation plan problem and in particular a volcano evacuation plan problem is reviewed. The complexity of evacuation plan problems is also considered. We give a procedure to construct the hazard zone background knowledge needed for reasoning about evacuation planning. Finally, this work is illustrated on the Popocatépetl case.

Chapter 8 presents the use of preferences in answer sets. It starts defining the problem of finding alternative evacuation plans in case part of the predefined evacuation plan becomes blocked. Then we show how this problem can be solved using two different approaches, i.e., CR-programs and PP language. This chapter also proposes an extension of $\mathcal{PP}$ language where propositional connectives and temporal connectives allow us to represent compactly preferences having a particular property. This chapter concludes with the presentation of a brief overview about the relationship between language $\mathcal{PP}$ and propositional Linear Temporal Logic (LTL) [21, 40]. The idea is that language PP could take advantage of the working framework of LTL to express preferences.

Chapter 9 introduces the notion of *Semantic Contents* of a program as an alternative point of view to obtain different answer set semantics of a program. In particular, we show how to obtain the *standard answer sets*, the *generalized answer sets*, the *minimal*

generalized answer sets and new answer set semantics introduced in this chapter called *partial answer sets*.

Finally the conclusions of this work are presented.