WebREG
Un outil interactif pour apprendre les expressions référentielles

Mehdi Yousfi-Monod

22 juin 2010

Table des matières

1 Introduction **2**
1.1 La génération de texte dans le CALL 2
1.2 La génération d'expression référentielles 3

2 WebREG : l'application **4**
2.1 Les entités, les propriétés et les relations 5
2.2 Les activités 6
2.2.1 Entraînement 8
2.2.2 Où est la cible ? 9
2.2.3 Quelle est la bonne ER ? 9

3 WebREG : la génération des ER **9**
3.1 Un algorithme par séparation et évaluation 9
3.2 L'algorithme adapté à WebREG 15
3.2.1 Les N meilleurs sous-graphes 16
3.2.2 L'ambiguïté de la cible et des repères 18
3.2.3 La fonction de coût 21
3.3 La réalisation linguistique 23

4 Perspectives **23**
4.1 Propriétés, relations, entités 23
4.2 Activités 24

4.3 Génération des ER 25

5 Conclusion **25**

A Annexes : organisation du code source et des données **28**
A.1 Côté client : scène interactive 28
A.2 Côté serveur : algorithme et réalisation 30
A.3 Les données 31

1 Introduction

Ce travail se situe dans le contexte de la Génération d'Expression Référentielles (**GER**) et chevauche partiellement le domaine de l'enseignement de langues assisté par ordinateur ou *Computer-Assisted Language Learning* (**CALL**). L'objectif est de produire une application informatique exploitant la GER pour aider un apprenant à comprendre comment désigner des objets (ou entités) à l'aide d'expressions référentielles (**ER**).

1.1 La génération de texte dans le CALL

L'objectif est de proposer à un apprenant une activité informatique interactive comme aide pour l'enseignement d'une langue. Dans notre travail, l'élément interactif est une application Web que nous avons développée et nommée WebREG. Pour le matériel à apprendre, nous nous concentrons sur une sous-tâche de la génération de texte : les expressions référentielles. Cette tâche correspond à une des étapes du processus de génération de texte proposé par Reiter et Dale ([8], pp. 49). Le tableau 1 présente la décomposition en modules et tâches proposée par les auteurs.

La génération des ER est située comme tâche de contenu et de microplanification (*microplanning*). Sur le plan de l'apprentissage humain, nous nous concentrons donc sur une des tâches requises d'un humain pour s'exprimer dans une langue. Notre application a pour but d'apprendre à un humain comment écrire une ER qui désigne une entité et la distingue des autres. Nous souhaitons proposer cet apprentissage pour plusieurs langues.

Module	Tâche de contenu	Tâche de structure
Macro-planification	1 : Choix du contenu	2 : Structuration du document
Micro-planification	4 : Lexicalisation **5 : GER**	3 : Aggregation
Réalisation	6 : Réalisation linguistique	7 : Réalisation structurelle

TABLE 1 – Les différentes tâches de la génération de texte selon le modèle proposé par Reiter et Dale [8].

1.2 La génération d'expression référentielles

La GER appartient au domaine de la génération automatique de textes (*Natural Language Generation* ou *NLG*). Une ER est une forme linguistique utilisée pour désigner une entité dans un monde réel ou imaginaire ([2], pp. 1). Dans ce document, nous traiterons des ER possédant les caractéristiques suivantes, similairement à [4] :

1. elles sont réalisées linguistiquement en groupes nominaux (p. ex. « la chaise »), plutôt que pronoms ou autres mécanismes linguistiques alternatifs utilisés pour la référence ;
2. elles font référence à des objets physiques, plutôt qu'à des entités abstraites ;
3. elles sont uniquement destinées à faire identifier un objet cible à un récepteur [1] (*hearer*), et ne sont pas destinées à satisfaire tout autre but communicatif.

En reprenant la terminologie de [4], l'objectif de notre GER est de satisfaire le but communicatif référentiel (*referential communicative goal*) [3] en générant une **description distinctive** (*distinguishing description*) de l'entité référée, ou entité cible, c'est-à-dire une description précise qui ne correspond à aucun autre objet de l'**ensemble du contexte** (*context set*). Cet ensemble est composé de toutes les entités sensées être présentement observées par le récepteur. Cet ensemble, s'il est privé de l'entité cible, est

1. La personne à qui l'ER est destinée est un généralement lecteur ou un auditeur, et plus particulièrement un apprenant dans notre cas.

appelé **ensemble de contraste** (*contrast set*) ou ensemble de **distracteurs potentiels** (*potential distractors*) par les auteurs.

Pour distinguer l'entité cible des autres, nous utilisons des attributs de l'objet correspondant regroupant des **propriétés** (taille, couleur...) et des **relations** avec les autres objets (à gauche de...). Le type d'objet (chaise, canapé...) est une propriété particulière, obligatoire pour la réalisation car elle devient la tête du groupe nominal. Nous notons ces attributs sous la forme de couples $\langle attribut, valeur \rangle$. Au niveau de la réalisation, ces propriétés et relations se traduisent en adjectifs et groupes prépositionnels modifiant le groupe nominal. Par exemple pour l'objet « chaise », la propriété $\langle couleur, rouge \rangle$ et la relation $\langle à\ droite\ de, bureau \rangle$, une réalisation possible est : « la chaise rouge à droite du bureau ». Le but de la GER est alors de sélectionner un ensemble de couples d'attributs et valeurs qui permet de distinguer l'entité cible.

Pour aider à déterminer quelles ER ne devraient pas être générées, les auteurs s'appuient sur les maximes de Grice (*Grice's Maxims* [6]) qu'ils interprètent de la façon suivante pour le contexte de la génération d'ER :

Qualité : une ER doit être une description précise du référent considéré ;

Quantité : une ER devrait contenir suffisamment d'information pour permettre au récepteur d'identifier l'objet référé, mais pas davantage ;

Pertinence : une ER ne devrait pas mentionner des attributs non discriminants, et donc qui n'aident pas à distinguer le référent considéré de l'ensemble de contraste ;

Manière : une ER devrait être courte lorsque c'est possible.

Le respect et l'interprétation de ces maximes dépend partiellement de l'objectif et du contexte de l'application. Nous en discutons après avoir présenté notre application, en section 3.

2 WebREG : l'application

WebREG est un logiciel reposant sur les technologies Web et proposant à un utilisateur plusieurs activités d'apprentissage basées sur la génération multilingue d'ER. L'application est accessible en ligne à l'adresse suivante :

`http://www-etud.iro.umontreal.ca/~yousfim/WebREG/`

L'interface propose une scène graphique en 2 dimensions incluant un ensemble d'entités représentées par des images. La figure 1 est une capture d'écran illustrant l'interface. Chaque entité possède un ensemble de propriétés et peut être mise en relation avec d'autres entités. Ces entités, leurs propriétés et l'image qui leur est associée ont été extraites du corpus d'expressions référencielles TUNA [2] [9] utilisé dans la compétition *TUNA Challenge* [3] qui traite spécifiquement des ER. La configuration de la scène, c'est-à-dire le choix des entités, de leur position et de leurs propriétés est soit définie par le programme soit par l'apprenant selon l'activité.

WEBREG peut générer une ou plusieurs ER pour une entité cible et une langue données, ce processus est décrit en section 3. Présentement, l'**anglais** et le **français** sont implantés pour la génération des ER. Le texte de l'interface est quant à lui uniquement en anglais.

2.1 Les entités, les propriétés et les relations

L'ensemble des entités contient toutes celles de la section mobilier (*furniture*) du corpus TUNA : une chaise, un bureau, un ventilateur et un canapé. Nous avons ajouté à cela des boîtes qui peuvent contenir un de ces objets. Les propriétés de ces entités incluent celles disponibles dans le corpus TUNA :

taille : grand, petit ;

couleur : rouge, noir, marron, gris, vert, bleu ;

orientation : de face ou de dos, tourné à gauche ou à droite.

Certains objets du corpus TUNA ne disposent toutefois pas de toutes les valeurs de chaque propriété. Les boîtes sont quant à elles toujours grandes, de face et de couleur rouge, vert ou bleu.

Les entités sont placées sur la scène 2D, elles disposent donc de coordonnées, et il est alors possible de les désigner en les situant sur la scène ou relativement à d'autres entités. Dans notre modèle, ceci se traduit en propriétés de positionnement absolu (la chaise du coin inférieur gauche) ou relatif (la chaise la plus en bas), ou en relations de positionnement entre entités (la chaise à droite du bureau).

Nous avons ainsi défini les propriétés supplémentaires suivantes :

positionnement absolu : coin supérieur gauche ou droite, inférieur gauche ou droite ;

2. `http://www.csd.abdn.ac.uk/research/tuna/corpus/`
3. `http://www.itri.brighton.ac.uk/research/genchal09/tuna/`

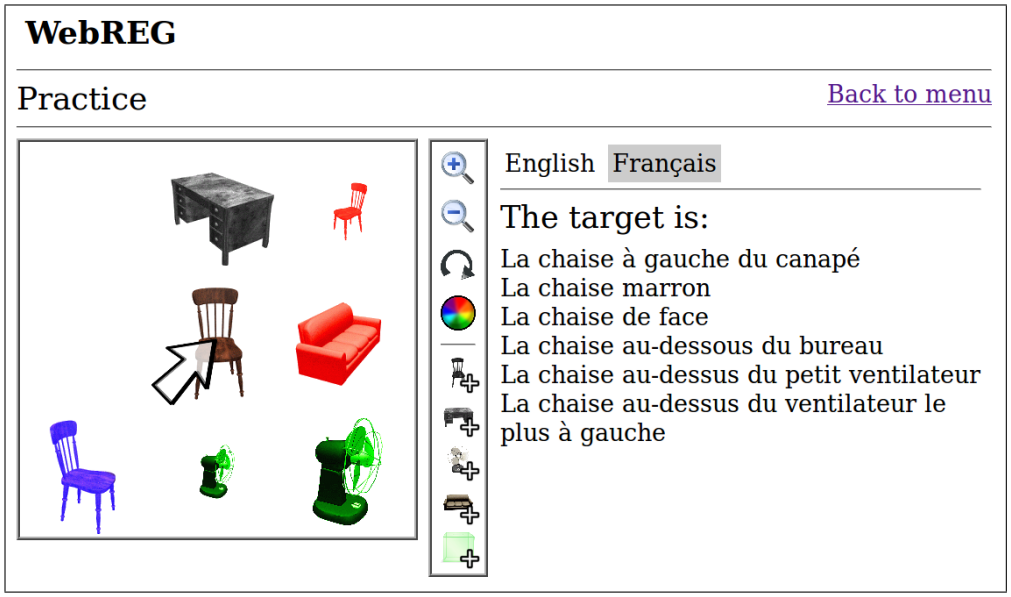


FIGURE 1 – Mode entraînement de WEBREG. À gauche se trouve la scène, au centre la boîte à outils, et à droite la liste des ER proposées pour la cible courante (désignée par une grosse flèche blanche) et les boutons de choix de la langue. L'apprenant peut modifier la scène en déplaçant les entités par glisser-déposer ou en utilisant la boîte à outils pour changer les propriétés des entités (taille, orientation et couleur) ou en ajouter de nouvelles. Une entité déplacée hors de la scène sera supprimée.

positionnement relatif : le plus haut, bas, à gauche ou à droite ;

et les relations suivantes :

positionnement : à gauche de , à droite de, au-dessus de et au-dessous de

contenance : dans (lorsqu'un objet est dans une boîte), contenant (lorsqu'une boîte contient un objet)

2.2 Les activités

Nous souhaitons solliciter l'apprenant sur la construction des ER par différentes interactions afin de varier les méthodes d'apprentissage. WEBREG inclut actuellement 3 types d'activités : un mode « *Practice* » (Entraînement), un exercice « *Where is the target ?* » (Où est la cible ?) et un exercice « *Which is the right Referring Expression ?* » (Quelle est la bonne ER ?).

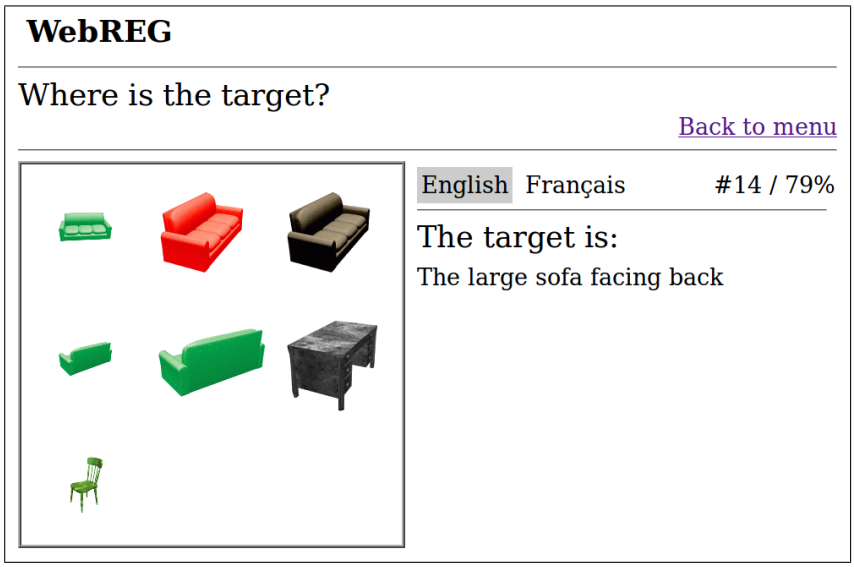


FIGURE 2 – Mode « Où est la cible ? ». L'apprenant doit cliquer sur l'entité qu'il pense être désignée par l'ER affichée. Dans le coin supérieur droit est affiché le nombre de parties jouées et le pourcentage moyen de réussite.

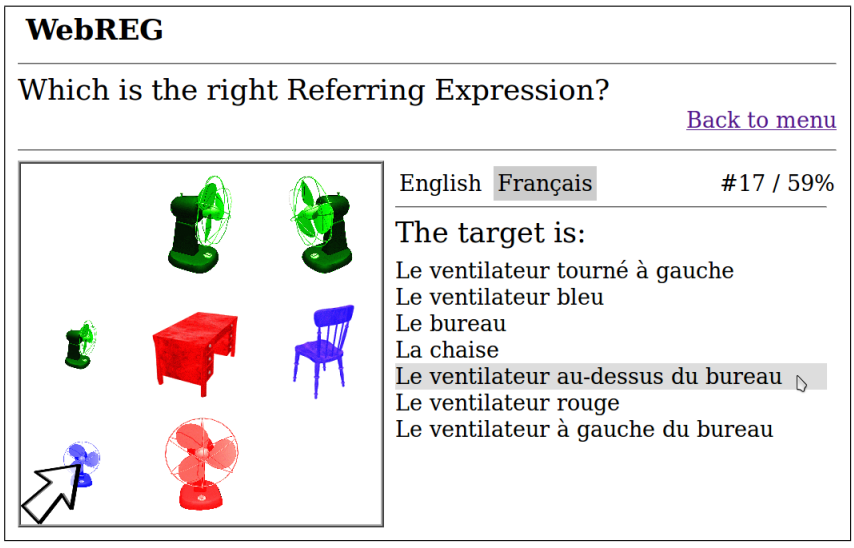


FIGURE 3 – Mode « Quelle est la bonne ER ? ». L'apprenant doit cliquer sur l'ER qui désigne l'entité cible (pointée par la grosse flèche blanche).

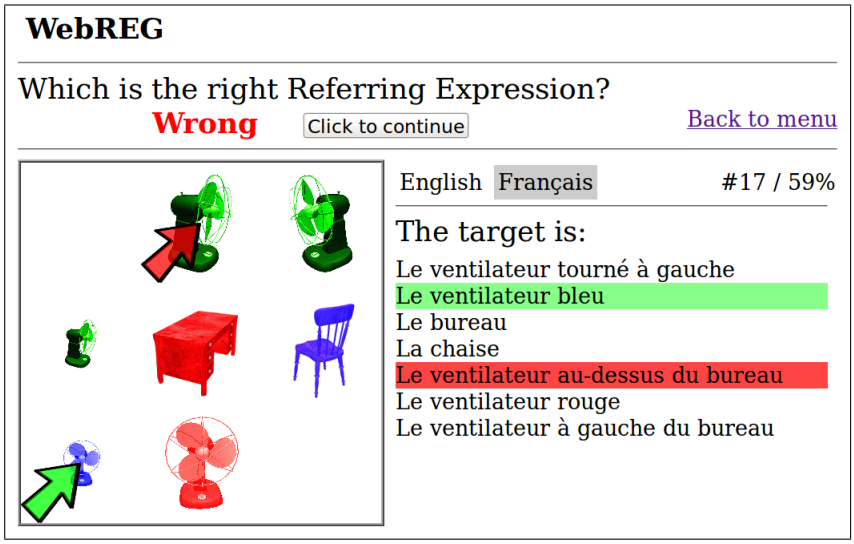


FIGURE 4 – Mode « Quelle est la bonne ER ? » à l'issue de la partie présentée dans la figure 3. Ici l'apprenant s'est trompé, le système lui montre la bonne réponse, en vert, et l'entité qui correspond à son mauvais choix, pointée par une flèche rouge.

À noter que quelle que soit l'activité, l'apprenant peut changer la langue de génération des ER à tout moment.

2.2.1 Entraînement

Ce mode permet à l'apprenant de manipuler la scène, sélectionner l'entité cible de son choix, et obtenir une liste d'ER valides la désignant. Sur la scène sont placées quelques entités, en nombre, position et attributs aléatoires. L'apprenant peut ensuite ajouter ou supprimer des entités, modifier leurs propriétés et positions, changer de cible. À chaque modification de la scène, un ensemble d'ER est généré par le système pour la cible en cours et dans la langue sélectionnée. Selon la scène, un grand nombre d'ER peut être généré, nous avons alors limité à un maximum de 10 pour le mode d'entraînement. Les ER sont présentées dans un ordre décroissant sur leur pertinence, selon l'estimation calculée par l'algorithme présenté en section 3.

La figure 1 offre une capture d'écran de l'interface de l'application pour le mode entraînement.

2.2.2 Où est la cible ?

Dans cet exercice, le système génère une scène avec plusieurs entités, choisit secrètement une cible, affiche son ER et demande à l'apprenant de désigner l'entité correspondante. L'apprenant désigne alors une entité en cliquant dessus, le système l'informe de son succès ou de son erreur, puis une nouvelle scène est générée et l'exercice recommence. Le nombre de parties jouées ainsi que le taux de réussite sont constamment affichés dans l'interface. La figure 2 présente une capture d'écran de cet exercice.

2.2.3 Quelle est la bonne ER ?

Cette activité propose un exercice symétrique au précédent : cette fois la cible est explicitement désignée, le système génère ensuite plusieurs ER, une seule correspond à la cible, les autres à d'autres entités de la scène. L'apprenant doit alors trouver quelle est l'ER correspondant à la cible. Ici aussi la scène est ensuite regénérée et le score comptabilisé.

Les figures 3 et 4 présentent deux captures d'écran de cet exercice, avant et après le choix de l'apprenant.

3 WebREG : la génération des ER

Pour générer une ER, WebREG doit être capable de sélectionner un ensemble de couples de $\langle propriété, valeur \rangle$ et/ou $\langle relation, valeur \rangle$ désignant l'entité cible de manière non ambiguë. Trouver la meilleure solution est un problème NP-difficile. Plusieurs approches proposent des heuristiques réduisant la complexité ou des optimisations réduisant l'espace de recherche. Dans le cadre de notre application, mettant en jeu peu d'entités et peu de propriétés et relations, nous avons opté pour un algorithme (section 3.1) qui parcourt systématiquement toutes les solutions possibles, jusqu'à une limite fixée par des contraintes psycho-linguistiques correspondant aux maximes de pertinence et de manière de Grice. Puis nous avons adapté l'algoritme aux besoins de notre application (section 3.2).

3.1 Un algorithme par séparation et évaluation

Pour développer l'algorithme correspondant, nous nous sommes basés sur l'approche de Krahmer *et al* décrite dans [7], qui définit un algorithme par

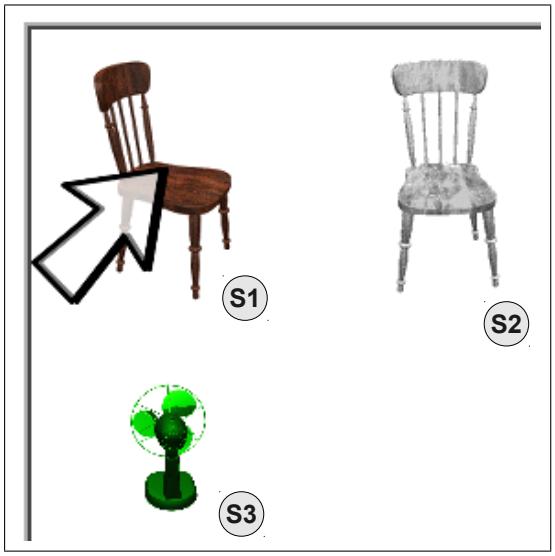


FIGURE 5 – Exemple d'une partie d'une scène, dans le coin supérieur gauche de la grille. L'identifiant interne de chaque entité est précisé dans un cercle placé à son côté.

séparation et évaluation (*branch and bound*) garantissant de retourner, si elle existe, une ER non ambiguë pour l'entité cible. Cet algorithme a été utilisé par le système décrit dans [1], qui a obtenu les meilleurs résultats [5] dans la compétition TUNA Challenge 2009, selon l'évaluation manuelle des systèmes.

Les auteurs représentent la scène par un graphe étiqueté orienté G, où les entités sont des sommets, et les propriétés et relations sont des arcs. L'étiquette d'un arc est la valeur de sa propriété ou relation. Une propriété est représentée par une boucle, c'est-à-dire un arc de l'entité vers elle-même.

Afin d'illustrer cette représentation des scènes par des graphes, nous présentons un exemple simple de scène dans la figure 5 et un extrait de son graphe correspondant dans la figure 6. Les propriétés de positionnement relatif et de placement dans les coins ont été volontairement omises dans le graphe pour des raisons de clarté. Le tableau 2 fournit toutes les valeurs des propriétés et relations de la scène de la figure 5. La scène est constituée de 3 entités, chacune correspond à un nœud du graphe : les sommets S1 et S2 sont des chaises, et S3 est un ventilateur. La chaise du coin supérieur gauche est l'entité cible, elle est représentée dans le graphe par un sommet composé d'un disque blanc dans un disque noir.

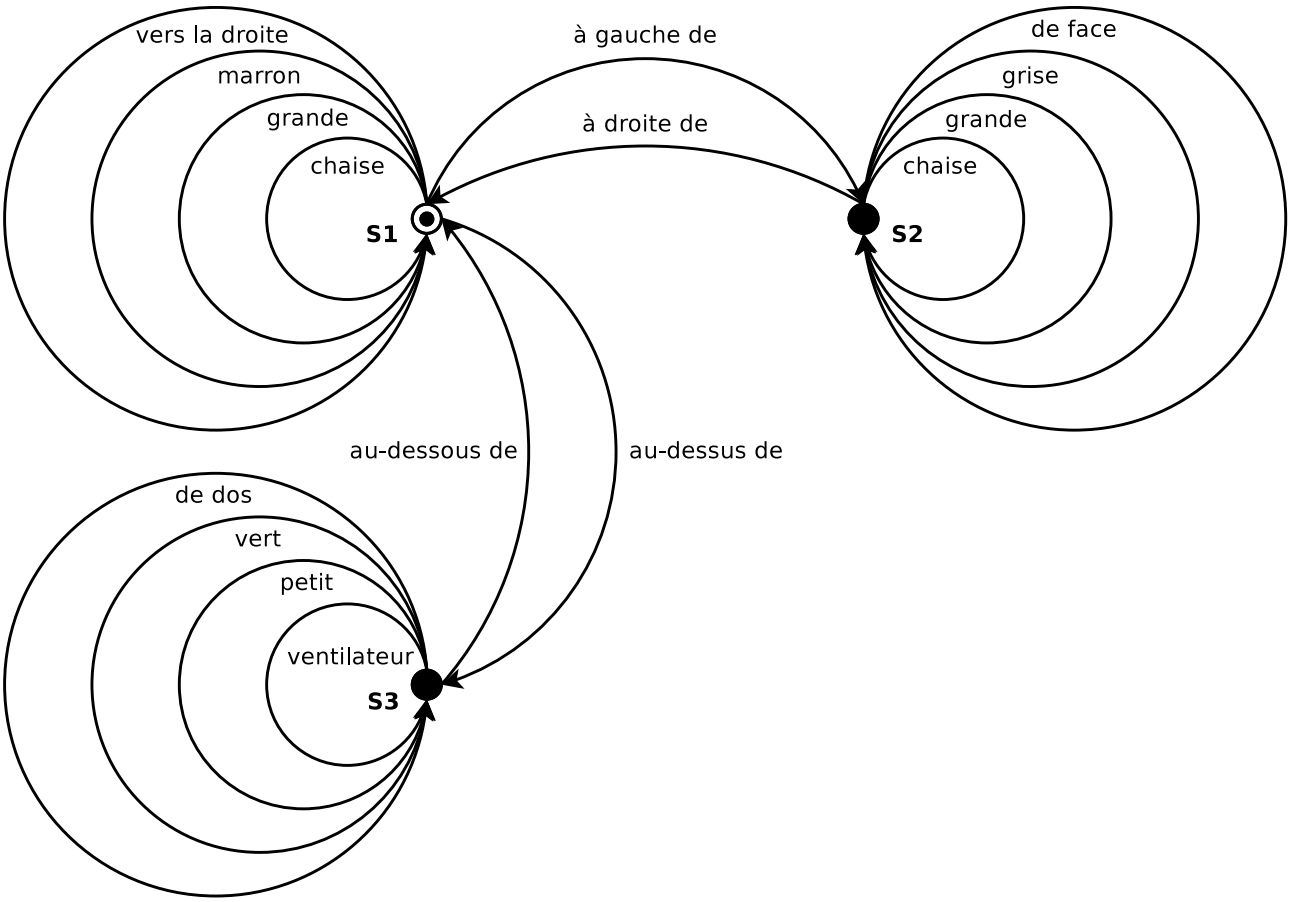


FIGURE 6 – Extrait du graphe correspondant à la scène de la figure 5.

	identifiant	S1	S2	S3
propriétés	**type**	chaise	chaise	ventilateur
	taille	grande	grande	petit
	couleur	marron	grise	vert
	orientation	vers la droite	de face	de dos
	position relative	la plus à gauche	la plus à droite	-
	coin	supérieur gauche	-	-
relations	**à gauche de**	S2	-	-
	à droite de	-	S1	-
	au-dessus de	S3	-	-
	au-dessous de	-	-	S1

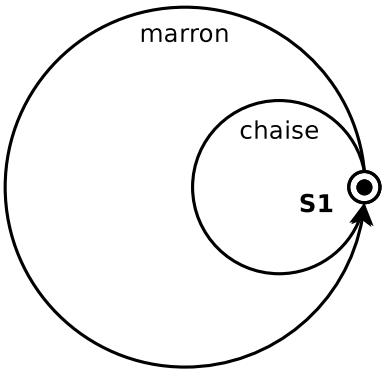
TABLE 2 – Identifiants, propriétés et relations des entités présentes dans la scène 5.

Dans cette section, nous reprenons la formalisation des auteurs pour présenter notre algorithme. À une scène donnée correspond un domaine fini d'entités D avec des propriétés P et des relations R. Soit $L = P \cup R$ l'ensemble des étiquettes (P et R disjoints). Alors $G = \langle V_G, E_G \rangle$ est un graphe étiqueté orienté, où $V_G \subseteq D$ est l'ensemble des sommets et $E_G \subseteq V_G \times L \times V_G$ est l'ensemble des arcs étiquetés.

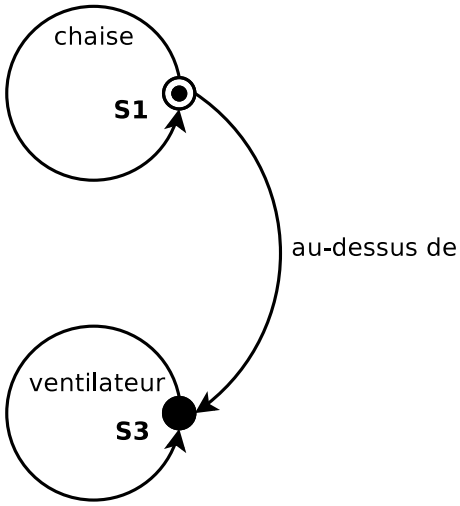
Selon cette modélisation, toute ER générable depuis la scène peut être représentée par un sous-graphe connexe H de G et un sommet v de H désignant l'entité cible. Une ER représentée par un graphe H est non ambigüe si et seulement si il n'existe aucun autre sous-graphe de G isomorphe à H. De tels graphes isomorphes à H peuvent être appelés distracteurs de H, à l'image de la nomenclature proposée par Dale et Reiter pour les entités.

Par exemple, les sous-graphes 7(a) et 7(b) de la figure 7 vérifient cette propriété, ils peuvent alors être respectivement utilisés pour générer les ER non ambigües suivantes : « la chaise marron » et « la chaise au-dessus du ventilateur ». Par contre les sous-graphes 7(c) et 7(d) sont isomorphes, l'ER « la grande chaise » est donc ambigüe.

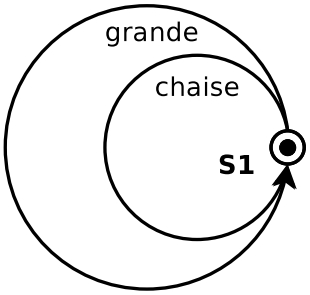
Le but est alors de construire progressivement un sous-graphe H à partir du sommet v, en lui ajoutant petit à petit des sommets et arcs de G, jusqu'à ce que plus aucun sous-graphe de G ne lui soit isomorphe. Toutes les constructions possibles de H sont testées recursivement. C'est ce que réalise



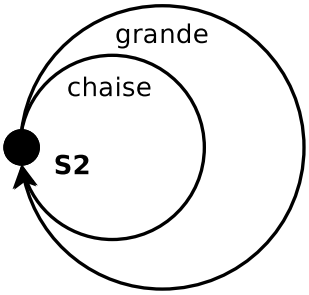
(a) La chaise marron



(b) La chaise au-dessus du ventilateur



(c) La grande chaise



(d) La grande chaise

FIGURE 7 – 7(a) et 7(b) sont des sous-graphes valides pour la GER non ambiguës pour la cible **S1**. 7(c) est isomorphe à 7(d), il ne peut donc pas permettre de générer une ER qui désigne l'entité **S1** de manière non ambiguë.

Fonction makeReferringExpression(v)

bestGraph $\leftarrow \perp$;
$H \leftarrow \langle \{v\}, \emptyset \rangle$;
retourner `findGraph(`*v, bestGraph, H*`);`

Fonction findGraph(v, bestGraph, H)

si *bestGraph* $\neq \perp$ **et** `coût(`*bestGraph*`)` $\leq$ `coût(`*H*`)` **alors**
    retourner *bestGraph*;
distracteurs $\leftarrow \{n \mid n \in V_G \wedge$ `matchGraphs(`*v, H, n, G*`)` $\wedge\ n = v\}$;
si *distracteurs* $= \emptyset$ **alors**
    retourner *H*;
pour chaque *arc* $e \in G$.`voisins(`*H*`)` **faire**
    $I \leftarrow$ `findGraph(`*v, bestGraph, H* + *e*`)`;
    si *bestGraph* $= \perp$ **ou** `coût(`*I*`)` $\leq$ `coût(`*bestGraph*`)` **alors**
        bestGraph $\leftarrow I$;
retourner *bestGraph;*

FIGURE 8 – Esquisse de la fonction principale *makeReferringExpression* et de la fonction de construction du sous-graphe *findGraph* de [7].

l'algorithme proposé par les auteurs de [7]. Dans le cas où il n'est pas possible de trouver un seul tel graphe, c'est-à-dire dans le cas d'une ER ambigüe, l'algorithme retourne le graphe vide.

Il existe souvent plusieurs ER non ambiguës pour désigner une entité, et donc plusieurs graphes H différents vérifiant la propriété d'isomorphisme. Pour choisir lequel sélectionner, l'algorithme inclut une fonction de coût d'un graphe, qui calcule une valeur de préférence de ce graphe, selon un poids attribué aux arcs. Les auteurs définissent les poids comme valant 1 pour tout arc du graphe et la fonction comme la somme du poids de tous les arcs du graphe. Cette fonction favorise ainsi les plus petits graphes, se conformant alors naturellement à la maxime de manière de Grice. La fonction de coût doit être monotone selon le nombre d'arcs afin de pouvoir arrêter la récursion dès qu'un graphe vérifie la propriété recherchée. En effet, toutes ses extensions seront plus coûteuses et donc moins intéressantes que ce graphe.

La figure 8 présente une esquisse des deux fonctions principales de l'algorithme proposé par les auteurs.

La fonction *makeReferringExpression*() est appelée avec le sommet

v correspondant à l'entité cible et doit retourner le meilleur sous-graphe *bestGraph* au sens de la fonction de coût *coût*(). *bestGraph* est initialisé au graphe vide, H au seul sommet v. Cependant, le sommet v ne peut produire une ER satisfaisante, la future réalisation linguistique requiert un nom, et donc le type de l'entité. Il est alors pratique d'ajouter l'arc correspond au type de l'objet à chaque ajout de sommet à H. Ainsi pour l'exemple de la figure 6, H sera initialisé au sommet **S1** plus l'arc *chaise*. Ce qui correspond, à ce stade, à l'ER « la chaise ».

La fonction récursive $findGraph()$ est ensuite appelée. La première condition d'arrêt teste si un meilleur graphe à été trouvé et s'il coûte moins que le graphe en cours de construction H. Si c'est le cas, alors inutile d'étendre d'avantage H, car le coût de toutes les extensions possibles sera forcément plus élevé de par la monotonie de la fonction de coût. Puis l'ensemble des distracteurs du graphe H est calculé, c'est-à-dire l'ensemble des sous-graphes de G qui lui sont isomorphes. La fonction *matchGraph* réalise cette tâche, nous ne la présentons pas ici, elle est expliquée dans [7]. Si aucun distracteur de H n'est trouvé, alors H vérifie la propriété recherchée, son ER sera donc non ambigüe, la fonction stoppe alors la récursion et retourne H.

Sinon, la méthode $G.voisins()$ calcule l'ensemble de tous les arcs e (propriété ou relation) de G qui peuvent être ajoutés à H tout en conservant sa connexité. Si e est un arc sortant vers un sommet non présent dans H, alors ce sommet est ajouté à H, ainsi que l'arc correspondant au type de l'objet, comme expliqué plus haut. $findGraph()$ est appelée sur chaque extension de H (H plus un arc e) et retourne leur meilleur sous-graphe I. Si I a un meilleur coût que le meilleur graphe actuel, alors I devient le meilleur graphe.

3.2 L'algorithme adapté à WebREG

L'algorithme précédent retourne un sous-graphe du graphe de la scène qui correspond à l'une des meilleures ER non ambiguës pour l'entité cible. Cependant, pour les objectifs de notre application, nous souhaitions d'une part pouvoir obtenir, selon l'activité, plusieurs ER pour une même entité (section 3.2.1), et d'autre part disposer d'un contrôle plus fin de l'ambiguïté de la cible et de ses repères (section 3.2.2).

FIGURE 9 – Conserver les N meilleurs graphes uniquement selon leur coût pourrait générer « la grande chaise » et « la grande chaise marron ». Un test d'inclusion de graphes permet d'éliminer la seconde ER.

3.2.1 Les N meilleurs sous-graphes

Nous souhaitions pouvoir proposer à l'apprenant plusieurs façons d'écrire une ER pour une entité donnée. Par exemple, dans la figure 5, les ER valides suivantes pourraient être générées : « la chaise marron », « la chaise tournée à droite », « la chaise du coin supérieur gauche », « la chaise la plus à gauche », « la chaise au-dessus du ventilateur ». Nous avions donc besoin d'obtenir plusieurs sous-graphes différents pour une même entité cible. Nous avons alors modifié l'algorithme pour qu'il conserve l'ensemble des N meilleurs sous-graphes. Nous appelons cet ensemble $NbestG$.

La fonction $findGraph()$ de l'algorithme original garantit de retourner le graphe correspondant à l'ER la moins chère selon la fonction de coût et non ambigüe, s'il en existe une. Ainsi, le graphe en cours de construction H est comparé au meilleur graphe actuel à l'aide de cette fonction. Cependant, nous souhaitons maintenant conserver un ensemble de meilleurs graphes, H doit alors être comparé à chaque graphe de cet ensemble. Si la fonction de coût est utilisée pour cette comparaison, alors $NbestG$ sera l'ensemble des N graphes les moins chers. Cet ensemble est proche de notre objectif, mais peut contenir des graphes incluant des arcs non discriminants.

Par exemple, si à la scène de la figure 5 nous ajoutons une troisième chaise partageant une propriété avec chacune des deux autres comme dans la figure 9, l'algorithme pourrait produire et conserver les deux sous-graphes correspondant aux ER suivantes : « la chaise marron » et « la grande chaise marron ». La première est issue de la construction de H où la propriété « marron » a été ajoutée en premier suffisant éliminer tous les distracteurs, alors que dans la seconde, c'est la propriété « grande » qui a été ajoutée en premier, éliminant la petite chaise, puis la propriété « marron », éliminant

l'autre chaise. Les deux ER ne sont pas ambiguës, cependant la seconde ne vérifie pas la maxime de pertinence.

Pour éviter que cela se produise, nous avons ajouté un test qui vérifie si le graphe courant H n'est pas inclus ou n'inclut pas un autre graphe de l'ensemble des $NbestG$. Si une de ces deux inclusions survient, nous ajoutons le graphe inclus (le plus petit) à $NbestG$.

La condition de rejet d'un graphe basée sur le coût a été alors supprimée L'ensemble $NbestG$ va petit à petit se remplir, puis c'est lorsqu'il atteint la taille N que nous utilisons la fonction de coût pour déterminer quel graphe doit être retiré de $NbestG$. La première condition d'arrêt basée sur le coût a donc été déplacée après les tests d'inclusions.

Traiter N meilleurs graphes plutôt qu'un seul augmente l'espace de recherche. Pour pallier aux pertes de performances associées, nous avons ajouté 2 nouvelles conditions d'arrêt :

- lorsque $H + e$ ne réduit pas le nombre de distracteurs par rapport à H (suivant la maxime de pertinence) ;
- lorsque $H + e$ génèrera une ER trop complexe pour être utilisée naturellement par un humain (suivant la maxime de manière). En pratique, nous limitons la taille des sous-graphes à un maximum de 2 sommets et autant d'arcs que nécessaire. Par ailleurs, un troisième sommet ne serait que très exceptionnellement sollicité compte tenu de l'ensemble des propriétés et relations disponibles dans notre application. Par exemple, la cible de la figure 5 pourrait premettre de générer l'ER peu claire « le ventilateur sous la chaise à gauche de la chaise grise », mais nous ne le permettons pas.

La figure 10 illustre les modifications apportées aux 2 fonctions pour gérer l'ensemble des $NbestG$. $bestGraph$ est remplacé par l'ensemble des $NbestG$, qui est passé par référence car il est global à tous les appels de $findGraph$. La fonction teste tout d'abord nos 2 premières conditions d'arrêt, puis si H n'a aucun distracteur, la fonction détermine s'il faut le conserver parmi les N meilleurs sous-graphes :

- les inclusions de graphes sont testées à l'aide de la méthode $H.estSousGrapheDe(H')$ qui retourne vrai si H est un sous-graphe de H', faux sinon ;
- s'il n'y a pas d'inclusion et qu'il y a moins de N meilleurs sous-graphes déjà conservés, alors nous ajoutons H à $NbestG$;
- sinon nous déterminons le graphe de $NbestG$ qui a le coût le plus élevé, à l'aide de la fonction $pire_{coût}$, et si H a un meilleur coût, alors il le

remplace, sinon il n'est pas conservé ;
– puis la récursion est stoppée.

S'il restait des distracteurs, alors les différentes extensions de H sont calculées et récursivement testées comme dans la fonction originale.

3.2.2 L'ambiguïté de la cible et des repères

L'algorithme ne produit aucun résultat s'il ne parvient pas à éliminer tous les distracteurs. Autrement dit, cela survient lorsque la configuration de la scène et les propriétés et relations disponibles pour les entités présentes ne permettent pas de produire une ER non ambiguë pour la cible. Dans notre application actuelle, ce genre de situation est plutôt rare, cependant nous pouvons trouver de telles situations. Nous pouvons aussi imaginer certains contextes pédagogiques limitant les propriétés et relations utilisables, ce qui nous confronterait plus souvent à ce problème.

Par exemple, la cible de la scène de la figure 11 pourrait être désignée par l'ER « le ventilateur le plus à gauche », mais si nous ne considérons pas les relations de positionnement relatif, nous ne pouvons plus produire un ER non ambiguë pour cette cible.

Il est toutefois possible de gérer de telles situations à l'aide d'ER indéfinies comme « un des ventilateurs », ce qui permet de retourner une expression valide en toute circonstance, offrant ainsi à l'utilisateur une information supplémentaire utile pour son apprentissage. Ainsi, afin de pouvoir gérer de tels cas, nous avons modifié $findGraph()$ pour qu'elle retourne les meilleurs sous-graphes, même s'ils souffrent de distracteurs. La fonction conserve en tout temps dans $minDist$ le nombre minimum de distracteurs rencontrés jusqu'alors et seuls les graphes qui ont le même nombre de distracteurs pourront être ajoutés à $NbestG$. Si un nouveau H réduit le nombre minimum de distracteurs, alors $NbestG$ est vidé et H devient son premier nouveau meilleur graphe. À l'issue de la récursion, $NbestG$ contiendra les N meilleurs graphes pour le nombre minimum de distracteurs trouvé.

De manière similaire, les repères, qui sont les sommets de H autres que celui correspondant à la cible, peuvent souffrir de distracteurs. L'algorithme original ne va cependant pas tenter de les désambiguïser, et si cet aspect n'est pas considéré, la génération d'une ER pour de tels graphes peut conduire à des expressions incorrectes. Par exemple la scène de la figure 12 peut conduire à la génération de l'ER « la chaise à gauche du canapé » alors qu'il y a deux canapés dans la scène.

Fonction makeReferringExpression(v, N)

$NbestG \leftarrow \emptyset$;
$H \leftarrow \langle \{v\}, \emptyset \rangle$;
`findGraph(`*v, NbestG, H, N*`)`;
retourner *NbestG;*

Fonction findGraph(v, &NbestG, H, N)

si *« H ne réduit pas les distracteurs »* **alors**
  retourner;
si *« H est trop complexe »* **alors**
  retourner;
$distracteurs \leftarrow \{n \mid n \in V_G \wedge$ `matchGraphs(`v, H, n, G`)` $\wedge\ n = v\}$;
si $distracteurs = \emptyset$ **alors**
  si $\exists H' \in NbestG \mid H$`.estSousGrapheDe(`$H'$`)` **alors**
    $NbestG \leftarrow NbestG - H' + H$;
  sinon si $\exists H' \in NbestG \mid H'$`.estSousGrapheDe(`$H$`)` **alors**
    $NbestG \leftarrow NbestG + H' - H$;
  sinon si $|NbestG| < N$ **alors**
    $NbestG \leftarrow NbestG + H$;
  sinon
    $H' \leftarrow$ `pire`$_{coût}$`(`$NbestG$`)`;
    si `cout(`H`)<cout(`H'`)` **alors**
      $NbestG \leftarrow NbestG - H' + H$;
  retourner;
pour chaque *arc* $e \in G$`.voisins(`H`)` **faire**
  `findGraph(`$v, NbestG, H + e, N$`)`;

FIGURE 10 – Modification de l'algorithme pour gérer les N meilleurs sous-graphes.

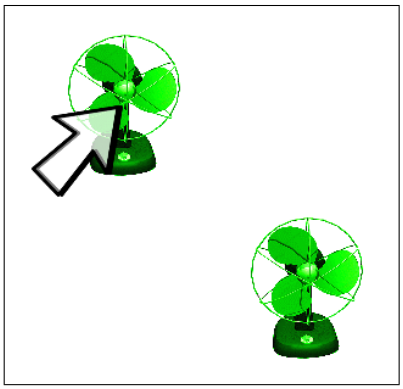


FIGURE 11 – Exemple de scène pouvant produire une ER ambiguë du type « le ventilateur » si nous n'utilisons pas les relations de positionnement relatif.

FIGURE 12 – Exemple d'ambiguïté possible pour un repère. Cette scène peut conduire à la génération de l'ER incorrecte « la chaise à gauche du canapé ».

Une première solution serait de générer une ER qui tient compte de l'ambiguïté du repère, avec un article indéfini : « la chaise à gauche d'un canapé ». Toutefois, même si le graphe produit suffit à identifier la cible de manière non ambigüe, l'ER générée n'est certainement pas celle qu'aurait produit un humain. Ce dernier préférera probablement ajouter une propriété au canapé pour le distinguer des autres, par exemple : « la chaise à gauche du canapé gris ». Nous avons alors modifié $findGraph()$ pour qu'elle ne stoppe pas la récursion si les repères ont des distracteurs, notamment si trop de sommets sont requis. Le graphe H est tout de même conservé au cas où il ne soit pas possible d'éliminer tous les distracteurs de ses repères. La première condition d'arrêt doit également prendre en compte ces nouveaux distracteurs.

Soit H' le graphe H dont les repères n'ont plus de distracteurs. H est alors un sous-graphe de H', car ce dernier s'est vu ajouter un ou plusieurs arcs. Cependant, notre algorithme actuel préfère les sous-graphes à leurs supergraphes qui sont éliminés. Nous avons alors pris en compte les distracteurs des repères dans les tests d'inclusion des graphes.

Enfin, dans le cas où un H n'est pas inclus, n'inclut pas et est pire que tous les graphes de $NbestG$, alors il est inutile de continuer la récursion, aucune de ses extensions ne sera meilleure que les N meilleurs graphes trouvés jusqu'alors, ceci étant garanti par la propriété de monotonie de la fonction de coût.

La figure 13 présente les modifications apportées à $findGraph()$ pour gérer ces ambiguïtés de la cible et de ses repères. Ces fonctions correspondent à la version actuelle de notre programme.

3.2.3 La fonction de coût

Notre fonction de coût d'un graphe se base sur celle suggérée dans [7] : le coût est la somme des valeurs de chaque arc. Nous avons fixée cette valeur à un. Par ailleurs, nous avons sensiblement modifiée cette fonction pour qu'elle pénalise les graphes :

- dont les repères ont des distracteurs ;
- contenant plusieurs fois le même type d'entité. Ceci car souhaitons éviter des ER peu naturelles comme « la chaise à gauche de la chaise rouge ».

Fonction makeReferringExpression(v, N)

$NbestG \leftarrow \emptyset$;
$minDist \leftarrow \infty$;
$H \leftarrow \langle \{v\}, \emptyset \rangle$;
`findGraph(`*v, NbestG, H, N, minDist*`)`;
retourner *NbestG;*

Fonction findGraph(v, &NbestG, H, N, &minDist)

si *« H ne réduit ni les distracteurs de la cible ni ceux des repères »* **alors**
  retourner;
si *« H est trop complexe »* **alors**
  retourner;
$distracteurs \leftarrow \{n \mid n \in V_G \wedge$ `matchGraphs(`*v, H, n, G*`)` $\wedge\ n = v\}$;
si $|distracteurs| < minDist$ **alors**
  $minDist \leftarrow |distracteurs|$;
  $NbestG \leftarrow \emptyset$;
si $distracteurs = minDist$ **alors**
  si $\exists H' \in NbestG \mid H$`.estSousGrapheDe(`*H'*`)` **alors**
    si *« H a moins de distracteurs de repères que H' »* **alors**
      $NbestG \leftarrow NbestG - H' + H$;
  sinon si $\exists H' \in NbestG \mid H'$`.estSousGrapheDe(`*H*`)` **alors**
    si *« H a plus ou autant de distracteurs de repères que H' »* **alors**
      $NbestG \leftarrow NbestG + H' - H'$;
  sinon si $|NbestG| < N$ **alors**
    $NbestG \leftarrow NbestG + H$;
  sinon
    $H' \leftarrow \texttt{pire}_{coût}(NbestG)$;
    si `coût(`*H*`)<coût(`*H'*`)` **alors**
      $NbestG \leftarrow NbestG - H' + H$;
    sinon retourner;
  si *« la cible et les repères n'ont plus de distracteurs »* **alors**
    retourner;
pour chaque *arc* $e \in G$`.voisins(`*H*`)` **faire**
  `findGraph(`*v, NbestG, H + e, N*`)`;

FIGURE 13 – Version finale de l'algorithme, incluant la gestion de l'ambiguïté de la cible et des repères.

3.3 La réalisation linguistique

Les N meilleurs sous-graphes construits sont utilisés pour générer les N expressions référentielles correspondantes. Le réaliseur actuel est très simple, il s'appuie sur un patron générique de syntagme nominal (SN) acceptant un déterminant, des adjectifs et des syntagmes prépositionnels (SP). Ces derniers sont composés d'une préposition et d'un syntagme nominal. Si plus d'un SP est présent pour un même SN, une conjonction de coordination « et » est insérée entre chaque SP.

Le lexique a été construit spécifiquement pour cette application, pour les entités, propriétés et relations traitées. Le lexique contient des informations sur les formes fléchies des lemmes, sur leur genre et nombre si nécessaire, et leur position dans le SN pour les adjectifs. Actuellement l'anglais et le français sont gérés, et l'ajout d'une langue syntaxiquement proche ne devrait pas poser de difficultés majeures.

Chaque sous-graphe en entrée est parcouru récursivement, à partir du sommet correspondant à la cible. La propriété `type`, qui spécifie le type d'entité, est convertie en un nom, les autres propriétés en adjectifs, et les relations en prépositions.

4 Perspectives

Cette section discute des perspectives envisagées pour les différentes parties de ce projet.

4.1 Propriétés, relations, entités

Nouvelles propriétés et relations. Les propriétés restent actuellement assez génériques et applicables à toutes nos entités. Ajouter de nouvelles pose la difficulté de pouvoir les contrôler dans l'interface de manière claire et efficace, sans trop complexifier l'interaction ou la partie algorithmique côté serveur. Pour les relations, d'autres pourraient être ajoutées si nous disposions d'une scène en 3 dimensions : « devant », « derrière ». Le principal obstacle est technique, car implémenter un environnement 3D pour une application Web requiert des technologies spécifiques et contraignantes (comme le langage Flash), et un temps de développement qui ne semble pas être disponible dans le cadre de notre projet.

Superlatifs et négation. Nous gérons partiellement les superlatifs, uniquement pour la position relative des entités. Ceci pourrait être généralisé à toute propriété, comme la taille (le plus gros) ou la distance (le plus près de). De même, le support de la négation augmenterait les possibilités de génération des ER (le ventilateur qui n'est pas dans la boîte).

Entités différentes Les activités actuelles n'utilisent que des entités de type mobilier. Les propriétés et relations ainsi que la réalisation actuelles sont limitées à ce domaine d'objets. Ajouter d'autres objets, mais aussi des animaux et des humains diversifierait considérablement l'apprentissage.

4.2 Activités

Autres types d'exercices. De nouveaux exercices pourraient augmenter progressivement la part de choix de l'apprenant dans l'identification ou la création d'une ER. Voici deux suggestions de tels exercices :

Validez l'ER Avant l'affichage de la scène, une cible est désignée, son ER est affichée, puis la scène est modifiée par le système avant d'être affichée. L'apprenant doit modifier la scène pour qu'elle valide l'ER affichée.

Créez la bonne ER Une cible est désignée puis le système propose un ensemble de mots suffisants pour que l'apprenant puisse construire une ER désignant la cible. L'apprenant procède par glisser-déposer en plaçant bout-à-bout des mots de l'ensemble.

Changement de langue. La langue peut-être actuellement changée à tout instant. Peut-être qu'il serait plus intéressant de contrôler dans quelles circonstances l'apprenant peut la changer.

Groupe de cibles. Actuellement la cible désigne une seule entité. Il pourrait être intéressant de proposer des exercices avec un groupe d'entités pour cible. La compétition TUNA Challenge 2009 considère de tels groupes pour une partie de son corpus.

Paramétrage des exercices. L'initialisation de la scène est actuellement assez simpliste. La qualité de l'apprentissage aurait certainement à gagner si la construction de la scène suivait un scénario pédagogique étudié et adapté

à chaque type d'exercice. Différents paramétrages, comme sur la difficulté de l'exercice, pourraient également améliorer l'apprentissage.

4.3 Génération des ER

Préférence lexicale. Les auteurs de [4] suggèrent qu'une ER devrait utiliser des classes lexicales basiques ou généralement préférées lorsque c'est possible. Ils définissent une nouvelle maxime qu'ils nomment « préférence lexicale ». Il serait alors intéressant que nous considérions de telles préférences, aussi bien dans la fonction de coût des sous-graphes qu'au niveau de la réalisation.

Les relations lexicales pourraient être utilisées pour modéliser ces préférences. Par exemple les relations d'hyperonymie et d'hyponymie peuvent aider à choisir entre « la chaise » et « le meuble ». En s'appuyant sur une ontologie prenant en compte de telles relations lexicales, il serait alors possible de mieux choisir les termes utiliser et d'offrir à l'apprenant un ensemble plus vaste d'ER générable. Par ailleurs les relations entre entités pourraient aussi bénéficier d'une telle ontologie. Par exemple le système pourrait choisir entre « au-dessous » et « sous ».

Réalisation : factorisation. Lorsqu'une entité est désignée par deux relations de même type, une factorisation pourrait être effectuée. Par exemple pour générer une ER comme « La femme à gauche des (deux) ballons ».

Réalisation : langues. Évidemment, à moyen terme, le support de nouvelles langues serait un plus considérable.

5 Conclusion

Ce travail est, à notre connaissance, le premier à s'appuyer sur une approche de génération des expressions référentielles pour proposer un système d'apprentissage de langue assisté par ordinateur. La sélection d'attributs discriminants de notre GER se base sur un algorithme qui a fait ses preuves dans la compétition *TUNA Challenge*. Nous avons adapté l'algorithme à notre tâche, afin qu'il puisse retourner plus expressions référentielles pertinentes pour une même entité, et qu'il puisse gérer l'ambiguïté de la cible et de ses repères de manière efficace et utile à l'apprenant. La base technique de WebREG, l'application prototypique résultante, est maintenant bien posée,

il reste principalement à étudier, développer et évaluer des scénarios pédagogiques pour obtenir une application réellement exploitable dans le cadre du CALL.

Références

[1] Ivo Brugman, Mariët Theune, Emiel Krahmer, and Jette Viethen. Realizing the costs : template-based surface realisation in the graph approach to referring expression generation. In *ENLG '09 : Proceedings of the 12th European Workshop on Natural Language Generation*, pages 183–184, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[2] Robert Dale. *Generating Referring Expressions : Constructing Descriptions in a Domain of Objects and Processes.* MIT Press, Cambridge, MA, 1992.

[3] Robert Dale and Nicholas Haddock. *Content determination in the generation of referring expressions.* Computational Intelligence, 7(4), 1991.

[4] Robert Dale and Ehud Reiter. Computational interpretations of the Gricean maxims in the generation of referring expressions. *Cognitive Science*, 19(2) :233–263, 1995.

[5] Albert Gatt, Anja Belz, and Eric Kow. The tuna-reg challenge 2009 : Overview and evaluation results. In *ENLG '09 : Proceedings of the 12th European Workshop on Natural Language Generation*, pages 174–182, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[6] Paul Grice. Logic and conversation. In Peter Cole and Jerry L. Morgan, editors, *Syntax and semantics*, volume 3. New York : Academic Press, 1975.

[7] Emiel Krahmer, Sebastiaan van Erk, and André Verleg. Graph-based generation of referring expressions. *Comput. Linguist.*, 29(1) :53–72, 2003.

[8] Ehud Reiter and Robert Dale. *Building natural language generation systems.* Cambridge University Press, New York, NY, USA, 2000.

[9] Kees van Deemter, Ielka van der Sluis, and Albert Gatt. Building a semantically transparent corpus for the generation of referring expressions. In *INLG '06 : Proceedings of the Fourth International Natural Language Generation Conference*, pages 130–132, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

A Annexes : organisation du code source et des données

Le code source est réparti dans plusieurs fichiers et plusieurs langages de programmation. Une partie est exécutée côté client (section A.1) et une partie par le serveur hébergeant l'application (section A.2). L'interactivité de WebREG ne peut être gérée que côté client, par le navigateur internet, et le reste de l'application pourrait certainement fonctionner côté client comme côté serveur. Nous avons fait le choix de ne pas gérer la génération des ER côté client pour plusieurs raisons :

- le langage Javascript n'est pas adapté aux processus requis : il aurait été moins pratique et efficace de créer et manipuler en Javascript les structures de données requises pour la partie algorithmique et la réalisation linguistique. Le langage PHP offre notamment un modèle de programmation par objets plus puissant et sécurisé ;
- l'interpréteur Javascript est contrainte par le navigateur : dans le cas où certains traitements algorithmiques pourraient requérir un temps de calcul dépassant quelques secondes, le Javascript ne serait pas adapté, car dans ces conditions le navigateur Internet peut arrêter le script ou demander à l'utilisateur s'il souhaite le laisser continuer ou non, ce qui n'est pas souhaitable dans le cadre de notre application ;
- la gestion des ressources est grandement facilitée côté serveur : les fichiers n'ont pas à être envoyés puis chargés côté serveur, leur format est plus libre et donc leur modification et leur chargement plus aisée.

A.1 Côté client : scène interactive

La scène est gérée côté client, et écrite dans les langages HTML, CSS et Javascript (JS).

Le code HTML contient uniquement le squelette de la page principale de l'application. Il consiste en un petit ensemble d'éléments `<div>` reflétant l'organisation visuelle de l'interface.

Le code CSS contient principalement des informations de taille, positionnement et visibilité des éléments. La visibilité est utilisée par exemple pour afficher ou non la flèche désignant la cible, la corbeille lorsqu'on glisse une entité hors de la scène, mais aussi le filtre désactivant l'interaction sur la grille

de jeu à la fin de chaque partie. Il peut être intéressant de modifier le code CSS si on souhaite changer les dimensions relatives, absolues ou minimum des champs principaux de l'interface (scène, langue, ER), ou encore les taille et position de la flèche de la cible, de la corbeille, ou des « plus » sur les boutons d'ajout d'entité.

Le code JS utilise la bibliothèque JQuery [4] qui propose une abstraction d'écriture efficace du code, de puissantes fonctions de manipulation des données Javascript, HTML et CSS, et une garantie de compatibilité [5] avec les principaux navigateurs Internet [6].

Le code JS est paramétré selon l'activité en cours. Il génère la scène, puis offre des moyens d'interaction à l'apprenant selon l'activité. Il s'appuie au maximum sur les propriétés CSS pour gérer l'affichage de tous les éléments HTML. En effet, beaucoup d'opérations consistent à ajouter ou enlever une classe HTML (attribut *class*) d'un élément de l'arbre DOM.

En plus de la bibliothèque JQuery, le code JS est réparti dans 4 fichiers [7], présentés ci-dessous.

WebREG-init.js définit la configuration initiale de la scène et associe les événements souris aux différents éléments HTML concernés. Par exemple, tout élément possédant la classe CSS *entity* récupère l'événement de pression du bouton gauche de la souris afin d'initialiser un glisser-déposer.

WebREG-entities.js définit la classe JS *entity* (au sens programmation objet) ainsi que quelques fonctions de manipulation des entités, comme l'ajout ou la suppression. Un objet entité est associé à l'élément HTML correspondant, possède un identifiant, un type d'objet (ballon, homme...), des valeurs de taille et de couleur, des coordonnées sur la grille de la scène, et un éventuel lien vers un conteneur ou un contenu si l'entité est dans une boîte ou est une boîte qui contient une autre entité. Un ensemble de méthodes permet de changer les propriétés de l'entité.

4. `http://jquery.com/`

5. Très pratique pour la gestion des événements qui fonctionne différemment selon les navigateurs, surtout avec Internet Explorer.

6. Les versions récentes de Firefox, Safari, Chrome, Internet Explorer et Opera.

7. Accessibles ici : `http://www-etud.iro.umontreal.ca/~yousfim/WebREG/js/`

WebREG-board.js gère les opérations liées directement à la grille de jeu. Principalement le glisser-déposer et les interactions des exercices. Les événements de pression du bouton de la souris, de déplacement de la souris et du relâchement du bouton sont liés à des fonctions de ce script. Le script s'assure par exemple que l'entité en déplacement ne survole pas une entité déjà sur la grille. Ou encore, si l'entité est déplacée hors de la grille, le script ajoute une classe CSS à l'entité pour faire apparaître la corbeille, et si l'entité est déposée à ce moment, le script la supprime.

WebREG-RE.js contient les fonctions de gestion des ER, principalement celle qui envoie une requête aux scripts PHP pour obtenir les ER réalisées. Par une technique de type Ajax [8], le code JS envoie la configuration de la scène à un script PHP (`GRE.php`), puis récupère le résultat et l'affiche dans l'interface.

A.2 Côté serveur : algorithme et réalisation

Le processus de génération de l'ER est gérée côté serveur, par un ensemble de 6 scripts PHP.

index.php est le premier script [9] chargé de WebREG, il génère le squelette HTML selon l'activité en cours, incluant le chargement des fichiers JS requis. Ce script commence par inclure le fichier `init.inc` qui charge le lexique utilisé pour la réalisation. Afin de ne charger qu'une seule fois le lexique par période d'activité de l'apprenant, une session PHP est démarrée, et le lexique est sauvé dans une variable de session.

GRE.php est un script [10] appelé par `WebREG-RE.js`, il récupère la configuration de la scène et retourne une liste d'ER, selon l'activité.

8. `http://en.wikipedia.org/wiki/Ajax_(28programming)`
9. Code source accessible ici : `http://www-etud.iro.umontreal.ca/~yousfim/WebREG/index.php?source`
10. Code source accessible ici : `http://www-etud.iro.umontreal.ca/~yousfim/WebREG/GRE.php?source`

init.inc est un sous-script [11] inclus par `index.php` et `GRE.php`. Il charge les autres fichiers d'extension `inc` qui contiennent les différentes classes utilisées par les scripts PHP, puis démarre ou recharge la session PHP. Si c'est la première exécution pour la session en cours, il charge le lexique.

lexicon.inc déclare les classes « Lemma », « NP » et « PP » utilisées pour charger le lexique [12] et effectuer la réalisation des ER. La fonction de chargement du lexique est également définie dans ce script.

entity.inc déclare la classe « Entity » qui peut être considérée comme une entité ou un sommet des graphes manipulés dans `graph.inc`. Un objet entité possède un identifiant, un ensemble de propriétés, de relations sortantes et de relations entrantes, et des méthodes pour ajouter une propriété ou une relation.

graph.inc déclare la classe « Graph », utilisée pour représenter la scène ou une partie de la scène dans une structure de type graphe. Un objet graphe possède un ensemble d'objets « Entity », l'identifiant de celle désignant le référent (la cible) dans la scène, et l'ensemble des méthodes utilisées pour construire le meilleur sous-graphe identifiant l'entité cible, notamment l'implémentation de l'algorithme présenté en figure 13. La fonction $makeReferringExpression()$ est implémentée dans la méthode $findGraph()$ et la fonction $findGraph()$ dans la méthode $recFindGraph()$.

A.3 Les données

WebREG utilise le lexique comme donnée textuelle et des images comme données graphiques. Les images sont contenues dans le dossier `img/`. Elles regroupent :

- les graphismes de l'interface : flèche, corbeille, boutons de la boîte à outils...
- les entités : toutes les images de grande taille du mobilier du corpus TUNA.

11. Les fichiers `inc` sont accessibles ici : `http://www-etud.iro.umontreal.ca/~yousfim/WebREG/inc/`

12. Contenu dans le fichier `lexique.txt` accessible ici : `http://www-etud.iro.umontreal.ca/~yousfim/WebREG/lingual/`

Les images des entités ont été converties depuis le format `gif` vers le format `png`, leur cadre rouge a été retiré et les formes détourées pour avoir un contour transparent. Pour les générer depuis les versions originales, nous avons utilisé l'outil `mogrify` de la suite d'outils libres `ImageMagick`[13], avec la commande Unix suivante :

```
mogrify -crop 450x450+2+2 -format png -alpha On -channel Alpha \
   -fill none -fuzz 8% -opaque white *Large.gif
```

13. `http://www.imagemagick.org/`