Sous-classes de formules propositionnelles à satisfiabilité tractable : solution

1. (a) Montrer que tout résolvant entre 2-clauses est une 2-clause.
   Si $C \vee A$ et $\neg A \vee C'$ sont deux 2-clauses, de résolvant $C \vee C'$, C et C' contiennent au plus un littéral chacune, donc $C \vee C'$ est une 2-clause.

   (b) On se donne un ensemble de 2-clauses S, de taille n. On pose $S_0 = S$, S_{n+1} l'ensemble des clauses de S_n plus tous les résolvants entre clauses de S_n, et $S_\infty = \bigcup_{n \in \mathbb{N}} S_n$. Montrer que le cardinal de S_∞ est borné par un polynôme en n.
   S_∞ ne peut contenir que des 2-clauses construites sur un vocabulaire d'au plus n variables propositionnelles, où n est la taille de S. En particulier, le cardinal de S_∞ est d'au plus $4n(n+1)/2 = 2n(n+1)$. ($O(n^2)$)

   (c) Proposer une réalisation de l'algorithme de résolution propositionnelle qui termine sur tout ensemble S de 2-clauses, et en déduire que 2-SAT est soluble en temps polynomial. (On ne demande pas un algorithme efficace.)
   Maintenir une liste de clauses pour représenter l'ensemble courant de clauses S_k. Pour calculer S_{k+1}, on calcule tous les résolvants entre clauses de S_k, et on rajoute uniquement les résolvants qui ne sont pas déjà dans S_k. Le nombre de clauses dans S_k est d'au plus $O(n^2)$, donc pour calculer S_{k+1}, on calcule jusqu'à $O(n^4)$ résolvants. Pour chaque résolvant généré, on teste s'il est dans la liste S_k en temps $O(n^2)$. (Noter que la comparaison de 2-clauses prend un temps constant.) On en déduit que S_{k+1} est engendrable en temps $O(n^6)$. D'autre part, on peut arrêter l'algorithme lorsque k atteint $2n(n+1)$, moment auquel plus aucune clause de S_∞ ne peut être rajoutée. Finalement, l'ensemble de 2-clauses S est satisfiable si et seulement si l'ensemble final $S_{2n(n+1)}$ de clauses ne contient pas la clause vide.
   On en déduit que cet algorithme de résolution, qui résoud 2-SAT par le théorème de complétude de la résolution propositionnelle, prend un temps $O(n^8)$, qui est polynomial. (L'algorithme de Even, Itai et Shamir [2] est essentiellement une version optimisée de cet algorithme, dans lequel les résolvants générés sont codés comme des chemins dans le graphe. Sa complexité est quasi-linéaire, par contre.)

   (d) L'algorithme de Davis-Putnam tourne-t-il en temps polynomial, exponentiel, ou autre sur les ensembles de 2-clauses? Commenter.
   Exponentiel : considérer les clauses $A_i \vee B_i$, $\neg A_i \vee B_i$, $A_i \vee \neg B_i$, $1 \leq i \leq n$. Aucune clause n'est ni pure ni unitaire ni tautologique, il est donc nécessaire d'effectuer une étape de splitting sur, disons, A_1. Ayant choisi une valeur de vérité pour A_1, il est de nouveau nécessaire d'effectuer un splitting sur, par exemple, A_2, et ainsi de suite, montrant que le temps pris par Davis-Putname est au moins $O(2^n)$ sur cet exemple de taille $6n$. Ceci démontre que l'algorithme de Davis-Putnam n'est pas toujours plus efficace qu'un algorithme de résolution.

2. (a) Montrer que si S ne contient pas de fait, alors S est satisfiable.

   Si S ne contient pas de fait, toutes les clauses de S contiennent au moins un littéral négatif. L'interprétation qui envoie toute variable vers $\perp$ (faux) satisfait donc S.

   (b) Montrer que si S ne contient pas de but, alors S est satisfiable.

   Si S ne contient pas de but, toute clause contient exactement un littéral positif. L'interprétation qui envoie toute variable vers $\top$ (vrai) satisfait donc S.

   (c) Déduire de la question 2a un algorithme en temps polynomial décidant HORN-SAT. (On utilisera un argument inspiré de ceux prouvant la correction de l'algorithme de Davis-Putnam.)

   D'après la question 2a, si S ne contient pas de fait, alors S est satisfiable, et l'algorithme s'arrête. Sinon, si S est satisfaite par une interprétation I, cette interprétation doit satisfaire tous les faits (de la forme A), et doit donc envoyer les variables A correspondantes sur $\top$ (vrai). On transforme alors S en remplaçant les variables A par vrai, c'est-à-dire en effaçant les clauses contenant A, et en effaçant $\neg A$ des clauses qui les contiennent, et l'on boucle. L'algorithme s'arrête, soit lorsqu'on a ainsi généré la clause vide (S est alors insatisfiable), soit lorsqu'il ne reste plus aucune clause (on a alors trouvé une interprétation satisfaisant la formule de départ).

   Le nombre de tours dans la boucle est d'au plus m, le nombre de variables de départ. À chaque tour, l'on collectionne par exemple dans une liste l'ensemble des faits de S, ce qui prend un temps $O(n')$, où n' est la taille de S, et l'on effectue les remplacements par vrais décrits ci-dessus en temps $O(n')$. Comme $m \leq n'$ et $n' \leq n$, où n est la taille de l'ensemble initial de clauses, on en déduit que l'algorithme tourne en temps $O(n^2)$. (L'algorithme de Dowling, Gallier et Minoux tourne en temps linéaire [1, 4].)

3. *L'idée des clauses Horn-renommables est due à Lewis [3].*

   (a) Soit $\mathcal{A}$ un renommage arbitraire. Montrer que S est satisfiable si et seulement si $R(S, \mathcal{A})$ l'est.

   Si S est satisfiable, disons par une interprétation ρ, alors il est clair que l'interprétation qui à toute variable A associe $\rho(A)$ si $A \notin \mathcal{A}$, et $[\![\neg A]\!]\rho$ sinon satisfait $R(S, \mathcal{A})$. La réciproque s'en déduit si l'on remarque que $R(R(S, \mathcal{A}), \mathcal{A}) = S$.

   (b) Soit C une clause quelconque. Pour chaque variable A apparaissant dans C, on crée une nouvelle variable propositionnelle x_A. Décrire un ensemble de 2-clauses $F(C)$ sur les variables x_A, tel que les interprétations qui satisfont $F(C)$ sont exactement les interprétations I telles que $R(C, \{A \mid I(x_A) = \top\})$ est une clause de Horn. (On note $\top$ le vrai.)

   Pour tout couple de littéraux L et L' fabriqués à partir de variables distinctes apparaissant dans la clause C, (ceci n'est applicable que si C contient au moins 2 littéraux) on exprime que L et L' ne peuvent pas être renommés chacun en un littéral positif :

   - *si $L = A$, $L' = A'$ sont tous les deux positifs, on génère la clause $x_A \vee x_{A'}$ (au moins une des deux variables doit être renommée);*
   - *si l'un exactement des deux littéraux est négatif, par exemple $L = \neg A$, $L' = A'$, on génère $\neg x_A \vee x_{A'}$ (si on renomme A, alors on doit renommer A');*

- *finalement, si les deux littéraux sont négatifs, soit $L = \neg A$, $L' = \neg A'$, alors on génère $\neg x_A \vee \neg x_{A'}$ (les deux variables ne doivent pas être renommées toutes les deux).*

L'ensemble $F(C)$ est alors l'ensemble de toutes ces clauses, lorsque L et L' parcourent l'ensemble des littéraux distincts de C. (Cet ensemble est donc vide si C est la clause vide ou une clause unitaire.)

Par construction, si $R(C, \{A \mid I(x_A) = \top\})$ est une clause de Horn, alors I satisfait F. Réciproquement, si I satisfait F, et si $\mathcal{A} = \{A \mid I(x_A) = \top\}$, alors $R(C, \mathcal{A})$ est par construction une clause C' dans laquelle, pour tous littéraux distincts L et L' de C', au plus l'un des deux littéraux est positif. On montre par récurrence sur le nombre de littéraux dans C' que C' est une clause de Horn. Si C' a au plus un littéral, c'est évident. Sinon, soit $C' = C'' \vee L'$, où L'' est un littéral de C' et C'' est la clause formée de tous les autres littéraux. Par hypothèse de récurrence, C'' est une clause de Horn. Si C'' est un but, alors C' est par définition une clause de Horn. Sinon, C'' contient un littéral positif L. Mais par hypothèse, il y a au plus un littéral positif dans $\{L, L'\}$: donc L' est négatif et C' est une clause de Horn.

(c) Quelle est la taille de $F(C)$ en fonction de celle de C ? En déduire, en utilisant les résultats de la question 1, que savoir si un ensemble de clauses S est Horn-renommable est décidable en temps polynomial. Semble-t-il faisable en temps polynomial de trouver un renommage s'il en existe un ?

La taille de $F(C)$ est $m(m-1)$, où m est la taille de C, puisque $F(C)$ contient $m(m-1)/2$ clauses à deux littéraux. Soit S un ensemble de clauses. S est Horn-renommable si et seulement si l'ensemble de 2-clauses $\{F(C) \mid C \in S\}$ est satisfiable : construire donc cet ensemble de clauses, en temps au plus quadratique ($O(n^2)$, où n est la taille de S), et décider sa satisfiabilité en temps $O(n^{2k})$, où k l'exposant trouvé en question 1 ($k = 8$ avec l'algorithme naïf par résolution).

Si $\{F(C) \mid C \in S\}$ est satisfiable, l'ensemble de tous les résolvants calculés ne contient pas la clause vide. Ceci ne permet pas de trouver en temps polynomial une interprétation satisfaisant l'ensemble de clauses $\{F(C) \mid C \in S\}$. En effet, considérons l'arbre de décision, tel que présenté dans le cours (Figure 2.6, p.45). Dire que l'ensemble de clauses est satisfiable, c'est dire qu'il y a une branche qui se termine sans passer par un nœud d'échec. La trouver peut prendre un temps exponentiel, l'arbre en question, même élagué aux nœuds d'échec, pouvant être de taille exponentielle.

(d) Adapter l'algorithme de Davis-Putnam (dans l'esprit de la question 2c) en un algorithme en temps polynomial décidant la satisfiabilité des ensembles S de clauses Horn-renommables. (On supposera qu'on a vérifié à l'avance que S était effectivement Horn-renommable.)

On parcourt les clauses unitaires L de S. S'il n'y en a pas, comme S est Horn-renommable, S est satisfiable. (En effet, S est satisfiable si et seulement si n'importe laquelle de ses versions renommées l'est.) Sinon, on applique la règle d'élimination des clauses unitaires de Davis-Putnam, ce qui réécrit S en un ensemble de clauses qui est encore Horn-renommable, et l'on boucle.

Le temps d'exécution de cet algorithme est le même que celui pour les ensembles de clauses de Horn, soit $O(n^2)$ pour la version naïve présentée ici, et $O(n)$ pour la version à base de graphes.

References

[1] W. F. Dowling and J. H. Gallier. Linear time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[2] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing*, 5:691–703, 1976.

[3] H. R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the ACM*, 25(1):134–135, 1978.

[4] M. Minoux. LTUR: A simplified linear-time unit resolution algorithm for Horn formulæ and computer implementation. Technical Report 206, Laboratoire MASI, CNRS UA 818, Université Pierre et Marie Curie, janvier 1988.