UCLouvain

epl

École polytechnique de Louvain

Exercices de circuits automatisés pour les cours d'électricité

Auteurs: **Arthur Billion, Arthur Detournay**
Promoteurs: **Olivier Bonaventure, Claude Oestges**
Lecteurs: **Bruno Dehez, Anthony Gego**
Année académique 2018–2019
Master [120] : ingénieur civil en informatique

Résumé

Ce mémoire couvre l'implémentation et l'utilisation de circuits électriques sur la plateforme INGInious dans le but d'améliorer la compréhension des étudiants. La technologie évolue tous les jours dans de nombreux domaines et l'éducation ne fait pas exception à la règle. Une partie du contenu des cours commencent à être en ligne pour proposer plus d'interaction avec l'étudiant. En effet, un contenu papier limite le nombre d'exercices et ne donne aucun retour sur le développement de ceux-ci.

INGInious est une plateforme en ligne capable d'évaluer des codes informatiques. Elle est actuellement utilisée à l'Université catholique de Louvain. Ce travail a pour but d'améliorer cette plateforme en permettant aux étudiants en électricité de s'entraîner et de recevoir des retours constructifs de leurs exercices. Cette extension a déjà été positivement critiquée par des étudiants de première et de seconde années à l'Université catholique de Louvain.

Remerciements

Nous aimerions remercier Olivier Bonaventure pour la supervision du mémoire et ses conseils.

Nous aimerions remercier Anthony Gégo qui nous a aidé à découvrir, comprendre et étendre la structure d'INGInious. Il nous a également aidés pour résoudre les diverses erreurs techniques émaillées tout le long du mémoire et nous a conseillés en termes d'approche.

Nous aimerions remercier Claude Oestges qui nous a aidés à fixer les objectifs de notre mémoire d'un point de vue de l'utilisateur. Nous aimerions également le remercier, lui, les assistants et les tuteurs du cours de circuits et mesures qui nous ont conseillés dans le but d'améliorer la plateforme.

Finalement, nous remercions Bruno Dehez qui a accepté de lire notre mémoire.

Sommaire

Chapitre 1

Introduction

L'automatisation est sûrement l'une des plus grandes avancées de ces deux derniers siècles. Les avancées dans ce domaine ont explosé dans une diversité de tâches qui nécessitaient précédemment l'intervention humaine. Des chaînes de montage aux voitures autonomes, les secteurs concernés sont innombrables et de nouveaux jalons d'innovation dans ce domaine sont atteints chaque jour.

La plateforme INGInious est une autre preuve de l'avancée de l'automatisation. L'outil INGInious a été créé afin de faciliter l'apprentissage des étudiants. Cet outil leur permet de s'exercer à réaliser diverses tâches. Des tâches liées aux connaissances requises durant les cours et/ou les travaux pratiques plus classiques organisés durant le cadre des études. Une fois la tâche accomplie, l'exactitude est vérifiée par la plateforme.

Si la vérification est déjà une tâche relativement complexe, ce n'est pas forcément suffisant pour rendre la plateforme intéressante pédagogiquement parlant. L'un des intérêts principaux est bien sûr d'assurer une correction automatique afin que l'étudiant puisse s'auto juger. Cependant, recevoir uniquement une indication binaire "vrai" ou "faux" n'aide pas à faire progresser l'étudiant. Le second intérêt dès lors, si l'étudiant se trompe, est d'automatiser un feedback, un indice qui sans donner la réponse, aide à lui faire réaliser son erreur. De telle sorte qu'il trouve lui-même la réponse correcte.

Du côté étudiant, INGInious a donc deux objectifs clefs : assurer la correction des exercices et aider l'étudiant en cas d'échec.

Dans un premier temps, INGInious était spécialisé dans la correction de code informatique dans des langages tels que C, Java, Python, etc. C'est pourquoi aujourd'hui, de nombreux cours d'informatique de l'Université catholique de Louvain utilisent la plateforme. Néanmoins, INGInious a un potentiel bien plus grand que le domaine de l'informatique. C'est dans cette optique que des extensions sont ajoutées à INGInious afin d'étoffer le panel des matières concernées.

Ainsi afin d'agrémenter INGInious, nous allons implémenter dans le cadre de

notre mémoire une extension permettant de s'exercer sur des circuits électriques. Notre travail vient donc se greffer au site déjà existant. Le premier but étant d'accompagner le cours **LELEC1370** - Circuits et Mesures électriques donné par le professeur Claude Oestges. Et notamment, de proposer de nombreux exercices d'analyse de circuit AC qui font intervenir des concepts comme les phaseurs et les nombres imaginaires. Ce type d'exercices en particulier, car ce ne sont pas des exercices qui nécessitent énormément de connaissances théoriques, mais qui requièrent de la rigueur et de l'expérience. Pour être à l'aise avec ces exercices, il faut surtout s'exercer encore et encore. Le souci principal actuellement étant que les étudiants manquent parfois de support pour s'entraîner ou pour vérifier leur réponse.

L'extension doit donc être capable de proposer des exercices clairs et limpides pour l'étudiant. Elle doit également donner une correction de l'exercice en prodiguer une aide si besoin à l'étudiant. Finalement, le recueil de questions doit être suffisamment étoffé pour que l'étudiant ait la capacité de s'entraîner autant de fois que nécessaire (et sans retomber sur des exercices déjà réalisés auparavant ce qui pourrait biaiser sa progression).

Ce mémoire présente premièrement l'état de l'art de notre mémoire: les outils utilisés, les solutions proches de notre problème et l'architecture d'INGInious.

Les fonctionnalités de l'extension seront ensuite présentées en décrivant l'interface utilisateur. Étant la première chose visible par l'utilisateur, une interface claire, lisible et qui permet une utilisation intuitive des outils, est un enjeu de l'extension développée. Cette partie explique ainsi les diverses facettes de l'interface et l'utilisation de la libraire Javascprit, MxGraph. Une librairie utilisée notamment sur des sites très connus tels que *draw.io*.

Ensuite, c'est l'utilisation de l'outil Spice capable de créer et de simuler un circuit qui est mise en avant. La correction des simulations vient ensuite s'y ajouter.

Afin de créer un recueil de questions inépuisables, le programme de génération aléatoire est détaillé. Ce programme est un outil qui, selon certains paramètres, permet la génération de circuits aléatoires et ainsi une multitude de questions potentielles.

Une correction binaire apporte peu d'information, donc comme mentionné plus haut, un feedback constructif est à apporter. C'est pourquoi, après la correction, les outils mis à la disposition de l'étudiant en vue de progresser seront explicités. En plus des outils, cette section aborde les différents messages que le correcteur automatique génère pour guider l'étudiant à trouver son erreur.

Finalement, l'avis des étudiants sur l'extension sera recueilli en vue de repérer et régler certaines failles et de trouver des pistes d'améliorations de l'outil.

Chapitre 2

État de l'art

2.1 Outils

L'un des points le plus importants de notre mémoire concerne bien évidemment, la simulation de circuits électriques. Le logiciel libre **Spice** est le programme tout désigné pour cette tâche. Ce logiciel permet notamment la simulation des composants électriques les plus courants (résistances, inductances, capacités) avec diverses méthodes d'analyse différentes :

- Courant continu
- Courant alternatif
- Transitoire
- Etc.

C'est pourquoi ce logiciel est utilisé dans ce mémoire et plus précisément son implémentation **NGSpice** afin de résoudre divers circuits électriques.

L'application propose de nommer les éléments du circuit et de former des équations. Former des équations implique de les résoudre et dans ce domaine la librairie Python **Sympy** est toute désignée. En effet, Sympy est une librairie entièrement écrite en Python et conçue pour le calcul symbolique. Ainsi, les outils de vérification des équations s'appuient sur Sympy.

Finalement, pour afficher les circuits interactifs, la librairie **mxGraph** est utilisée.

2.2 Solutions proches

D'autres plateformes en ligne permettent de simuler des circuits électriques comme CircuitLab [1]. CircuitLab permet de réaliser et simuler des circuits directement

à partir d'un navigateur web. L'éditeur permet de facilement créer un circuit en très peu de clics. De plus, de nombreux exemples de circuits à tester sont déjà préconçus. Dans un autre style de simulateur en ligne, il existe également DCACLab [2]. Comme CircuitLab, DCACLab, Solve elec propose un espace de laboratoire virtuel afin d'apprendre les concepts de l'électricité par l'interactivité. La plus grande différence entre les deux plateformes vient du côté visuel. En effet, CircuitLab propose une interface sobre, les circuits sont représentés de manière schématique là où DCACLab montre un interface plus colorée et où les circuits ressemblent plus à ce qu'on pourrait retrouver sur une platine d'expérimentation réelle.

Un autre point est la plateforme utilisée pour proposer des questions et des exercices. Le travail produit repose entièrement sur la plateforme INGInious de l'Université catholique de Louvain. INGInious est un site permettant d'acquérir de nouvelles compétences et qui propose des exercices, correctifs et feedbacks adaptés. Actuellement, la plateforme est essentiellement consacrée à l'informatique et aux diverses compétences en programmation. Un exemple de concurrent est le site Stepik [3]. La différence avec INGInious est la liberté de création des exercices. INGInious est beaucoup plus permissif que son concurrent, spécialisé dans des exercices précis.

Une multitude de sites proposent également de nombreuses leçons sur la physique des circuits électriques par exemple : Mastering Physics[4]. Malheureusement, si le nombre d'exercices parcourt un large spectre des exercices possibles, la quantité est toujours limitée. De plus, la plupart des sites ne proposent aucun feedback constructif et permet, dans le meilleur des cas, de disposer d'une résolution détaillée de l'exercice. L'idéal étant bien entendu de proposer un nombre d'exercices de circuits illimités et de prodiguer à l'utilisateur des outils lui permettant d'exposer sa résolution. Mastering Physics propose quand même des feedbacks constructifs, mais générés manuellement sans aucune automatisation.

Notre application se distingue de ces solutions, car non seulement nous proposons la simulation de circuits électriques, mais aussi un retour constructif sur le travail de l'étudiant.

2.3 Architecture d'INGInious

INGInious [5] est un projet open-source développé par la faculté INGI de l'Université catholique de Louvain initialement prévu pour tester et exécuter du code non fiable. L'outil est écrit en Python, repose sur Docker pour procurer un espace d'exécution sécurisé et sur MongoDB pour garder en mémoire les anciennes soumissions.

L'architecture d'INGInious [6] est séparée en trois : le backend, l'agent du backend et le frontend.

Le backend reçoit la soumission de l'étudiant et l'envoie à son agent qui est en charge d'exécuter le travail et qui interagit avec Docker pour démarrer de nouveaux containers. Les containers Docker sont des petits OS virtuels qui permettent une isolation entre les processus et les ressources du système d'exploitation hôte. Docker permet de créer et de gérer n'importe quel programme de n'importe quelle distribution Linux. Le container fait des vérifications sur la soumission reçue et retourne un des quatre statuts possibles : *success, failed, timeout ou crash.* L'agent se charge de renvoyer cette note au backend.

Le frontend sert d'interface web au backend. Il procure une interface simple, mais efficace aux étudiants et aux professeurs. Il est conçu pour être "sans état", le tout étant stocké dans une base de données permettant de répliquer le frontend horizontalement.

INGInious fournit un système simple de plugin [7] qui permet d'enregistrer des "hooks" pour étendre des fonctionnalités d'INGInious, ajouter des pages à l'interface, etc. Une liste de "hook" permet de générer tout type d'actions utiles selon des événements liés à INGInious.

INGInious propose aussi des statistiques à propos des étudiants. Le nombre d'exercices essayés, réussis et un aperçu du travail remis.

2.4 Structure des exercices

Une extension d'INGInious fonctionnelle est inutile sans ces exercices adéquats. Chaque exercice INGInious a son propre dossier qui doit posséder deux fichiers obligatoires pour fonctionner :

- Le fichier *task.yaml* décrit tout ce que INGInious a besoin de savoir pour vérifier la réponse de l'étudiant. Pour les exercices électriques, ce fichier précise l'environnement avec lequel travailler. Cet environnement correspond au conteneur docker *Ngspice* où le simulateur Ngspice et la bibliothèque Sympy sont installés. Ce fichier définit aussi l'exercice en lui-même, le circuit sous forme **XML** ainsi que les options de générations aléatoires (nombre de cycle, type d'exercice à générer) y sont enregistrées.

- Le fichier *run* est un script bash lancé lors des soumissions de l'étudiant. Il récupère d'abord le circuit contenant les réponses de l'étudiant sous forme **XML** et les équations de l'étudiant. Il coordonne ensuite l'utilisation des différents parsers (*parser_ngspice*, *parser_xml* et *parser_equation*) et l'utilisation de la simulation ngspice. Finalement, il retourne le feedback de l'étudiant en fonction des résultats des parsers.

Pour notre outil, les différents parsers utilisés par le fichier *run* sont aussi présents dans le dossier de l'exercice. Chaque dossier d'exercices contient donc cinq fichiers pour fonctionner correctement.

Chapitre 3

Interface

L'interface est la première chose vue par l'utilisateur. De plus, toutes interactions entre utilisateur et application passent par cette interface. C'est pour ces raisons qu'elle doit être claire et intuitive pour ne pas frustrer l'utilisateur. Malheureusement, si la forme n'est pas de qualité suffisante, les utilisateurs vont se désintéresser, peu importe le fond. Ainsi l'un des points clef du mémoire était de respecter ces contraintes.

Le plugin reposant sur INGInious, la base principale de l'interface repose sur tout ce qui avait déjà été créé préalablement. Le principal avantage est l'interface du site déjà construite et la familiarité de l'étudiant de l'UCL (utilisateur futur le plus probable) avec celle-ci. Ainsi, pour l'application, il y a deux éléments absents de base qu'il faut implémenter:

1. L'environnement des exercices. L'endroit où l'internaute prend connaissance de la question (du circuit) et y répond. À ce même endroit, il peut également utiliser divers outils afin d'exposer sa résolution.

2. La page d'édition. L'endroit où l'administrateur crée l'exercice de toute pièce.

3.1 Résolution

Lorsque vous choisissez un exercice de l'application, vous arrivez sur une page comme celle présentée sur la figure 3.1.

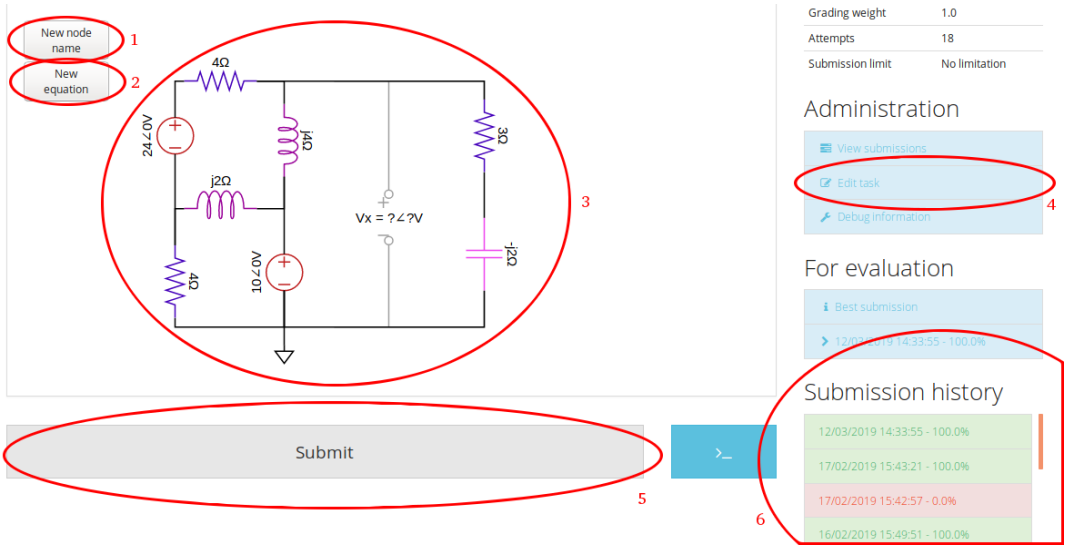
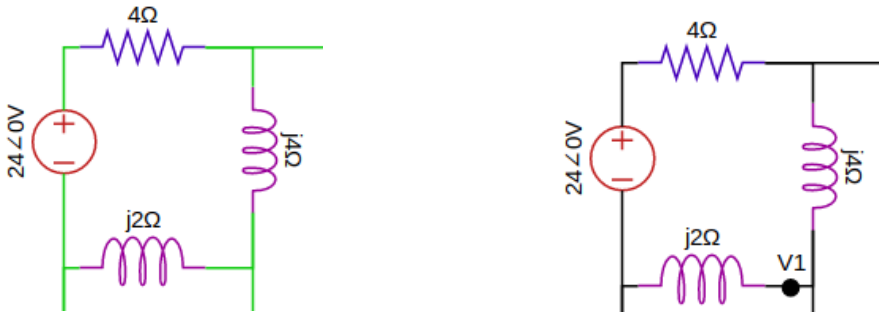


Figure 3.1: Page de résolution de l'exercice

Les fonctionnalités entourées en rouge sur cette figure, sont détaillées ci-dessous:

1. Le bouton *New node name* permet d'ajouter un label de tension pour pouvoir utiliser par la suite ce nom de tension dans les équations. Un exemple est fourni à la figure 3.2, après avoir cliqué sur le bouton, les fils deviennent verts pour montrer que l'on peut cliquer dessus pour ajouter un nouveau label de tension (ici $V1$). Avec cette manière de faire, un nouveau nom de noeud peut être ajouté n'importe où sur les fils. Ceci permet d'avoir une grande liberté pour écrire les équations du circuit.



(a) Fil cliquable (b) Nouveau label $V1$

Figure 3.2: Ajout d'un label de tension

2. Le bouton *New equation* permet d'ajouter une nouvelle équation (expliqué plus en détail dans la section feedback). Après avoir cliqué sur ce bouton, un

nouveau champ comme sur la figure 3.3 apparaît. L'étudiant pourra alors écrire son équation en utilisant des noms de label. Un bouton *Remove* et ∠ sont également présents pour respectivement supprimer le champ d'équation et ajouter le symbole de phase dans l'équation pour plus de lisibilité.

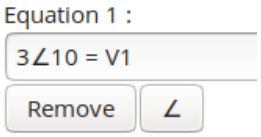


Figure 3.3: Equation

3. Le circuit électronique est interactif, la plupart des éléments peuvent être sélectionnés pour ajouter un label de courant comme montré sur la figure 3.4. Si la case *add name* est cochée, un label de courant (ici $I1$) ainsi qu'une indication de direction (->) sont ajoutés. Il est aussi possible de changer la direction du courant en cliquant sur le bouton *Switch direction*. Les noms de label ajoutés sont évidemment utilisables dans les équations.

(a) Élément cliquable

(b) Nouveau label $I1$

Figure 3.4: Ajout d'un label de courant

Le circuit peut contenir plusieurs types de questions, chaque question est représentée par un composant qui n'influence pas le circuit. Trois types de questions sont possibles comme on peut le voir sur la figure 3.5a : une question où une tension $V0$ entre deux bornes est demandée, une question où un courant $I0$ dans un fil est demandé et finalement, une question où il faut retrouver la valeur d'une impédance $Z1$ à partir d'une mesure de tension ou de courant du circuit (les "?" sont alors remplacés par la valeur de la mesure). Ces composants sont cliquables et une fenêtre *Answer* apparaît pour encoder la solution value∠phase.

(a) Trois questions possibles

(b) Encoder la réponse

Figure 3.5: Réponse aux questions

4. Le bouton *Edit task* est uniquement visible pour les utilisateurs possédant le droit de modifier ou d'ajouter des exercices. Il permet de passer à l'interface de création qui sera expliquée en détail dans la section suivante.

5. Une fois les équations et la solution encodées, le bouton *Submit* permet de soumettre l'exercice qui va alors être simulé et vérifié. La simulation et la vérification du circuit sont expliquées dans les sections suivantes.

6. Il est possible de revoir toutes les soumissions de l'exercice dans l'espace *Submission history*.

3.2 Création

Pour ajouter ou modifier un exercice électrique, il faut utiliser l'interface de création qui est dans l'onglet *Subproblems*. L'interface possède une boite à outils entourée en rouge sur la figure 3.6. Tous les composants de l'application se trouvent dans cette boite à outils : les impédances (résistance, inductance et capacité), les sources de courant et de tension commandées ou non, les labels pour indiquer une mesure, l'impédance cachée et finalement la masse. Chacun de ces composants peut être glissé de la boite à outils à la grille en maintenant le clic sur le composant et ensuite en le lâchant où l'on veut sur la grille (le bouton *Generate circuit* sera expliqué ultérieurement dans la section génération aléatoire).

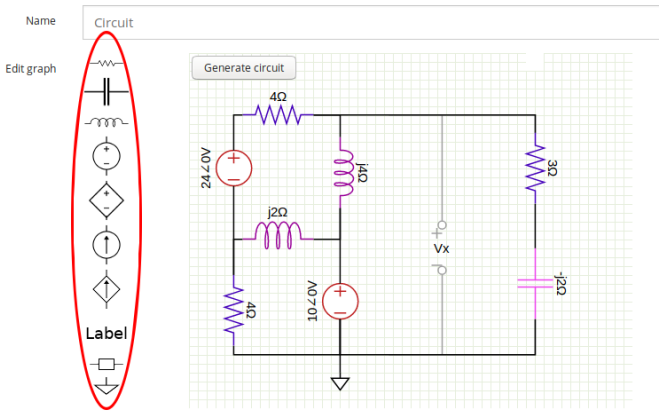


Figure 3.6: Boite à outils

Une fois le composant lâché sur la grille, ses paramètres pourront être encodés en le sélectionnant comme montré à la figure 3.7a. Les paramètres sont en général intuitifs : le nom, l'amplitude et la phase du composant. Cependant, les sources commandées et les labels possèdent des paramètres différents.

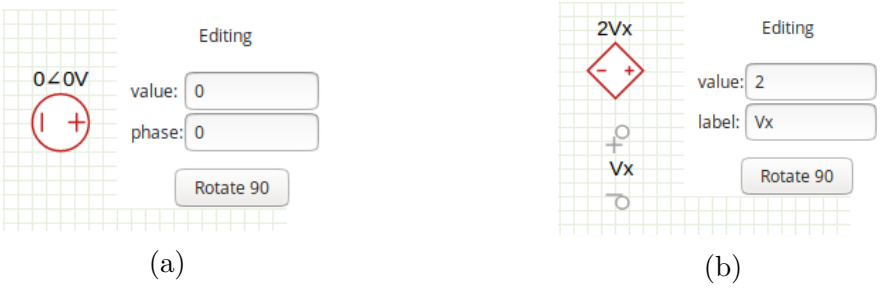
Les sources commandées sont définies grâce à un facteur et au nom de la mesure dépendante comme on peut le voir sur la figure 3.7b. La mesure dépendante est un label de tension ou de courant précédemment créé. Le label peut être de tout type: une question, avec des valeurs imposées ou aucun des deux.

Les labels possèdent un autre bouton *Switch type* qui permet de passer d'un label de courant à un label de tension ou vice versa. Les labels de tension doivent être placés en parallèle alors que les labels de courants doivent être mis en série. Trois cases pouvant être cochées sont présentes :

- *Question* indique si le label doit être une question de l'exercice.
- *Set* permet d'indiquer une valeur à la mesure pour faire des exercices inversés avec une impédance cachée. Attention, la valeur indiquée doit être correcte pour permettre à l'étudiant d'obtenir la bonne réponse. Une amélioration possible serait de simuler le circuit pour automatiquement encoder la valeur de la mesure.
- *Phase* indique si la phase du label est à prendre en compte ou non. Ceci permet de faire des exercices simplement résistifs où les phases ne sont pas utilisées.

Si aucune des deux cases *Question* et *Set* n'est cochée, le label nomme simplement une mesure du circuit pour être par exemple utilisé par une source commandée. Par défaut, les cases *Question* et *Phase* sont cochées étant le réglage le plus commun.

Les composants peuvent aussi être tournés en appuyant sur le bouton *Rotate 90* et supprimés en appuyant sur la touche *Delete* de l'ordinateur.



(a) (b)

Figure 3.7: Édition de composant

Pour ajouter les fil du circuit, les composants possèdent des ports qui s'illuminent en vert lorsque l'on passe la souris dessus. Il suffit ensuite de maintenir le clic sur un port et de lâcher le clic sur un autre port pour connecter deux composants ensemble. Cette manipulation est illustrée à la figure 3.8. La connexion entre fil n'est pas prise en compte dans l'application. Pour connecter un composant à plusieurs autres composants, il faut les connecter à partir des ports.

Figure 3.8: Création des fils

Les fils peuvent être modifiés après être connectés à deux composants. Tel que montré sur la figure 3.9, sélectionner les fils, permet de déplacer les carrés verts et ainsi de déplacer les fils. On peut aussi supprimer le fil en utilisant la touche *delete* de l'ordinateur.

Figure 3.9: Édition des fils

Attention à bien mettre au moins une source et une masse dans le circuit sinon la simulation avec Spice ne fonctionnera pas et l'exercice ne sera donc pas fonctionnel. Une amélioration possible serait d'empêcher l'utilisateur de sauvegarder les modifications de l'exercice si la masse et au moins une source ne sont pas présentes. Pour finir, il faut enregistrer les modifications en appuyant sur le bouton *save changes*.

3.3 Implémentation

L'interface utilisateur a été implémentée en JavaScript avec l'aide de la librairie mxGraph. MxGraph permet de créer des graphes interactifs avec beaucoup de liberté. Le célèbre site de création, édition et exportation de diagrammes draw.io est notamment basé sur cette librairie. Cet outil est donc très versatile et puissant. MxGraph est donc évidemment capable de créer toutes sortes de graphiques comme des diagrammes circulaires ou des histogrammes, mais aussi des graphiques plus complexes comme des circuits électriques. Cet outil nous permet de réaliser tout ce qui est nécessaire au plugin, mais il apporte aussi une complexité et donc un temps d'apprentissage de la librairie.

Le circuit électronique est composé de composants électriques tels que les résistances, les inductances ou encore les sources. Chacun de ces composants est un noeud du graphe et comprend :

- Une forme provenant d'un ensemble de pochoirs (stencils) qui sont chargés au lancement de l'application. Le pochoir, en plus de définir la forme du composant, définit aussi les noms et les emplacements des ports c'est-à-dire les différentes connexions du composant. Les pochoirs sont stockés dans un format **XML**.

- Un label représentant le nom, la valeur ou les valeurs en fonction du type de composant. Si la phase est égale à zéro, elle n'est pas affichée pour permettre aux étudiants de s'exercer sur des exercices plus simples (uniquement résistifs). Le nom du composant doit être unique pour respecter les préconditions de Spice. Pour ce faire, des compteurs pour chaque composant sont utilisés. Le label peut aussi contenir le nom et la direction du courant passant dans le composant.

- Une couleur qui donne un peu de relief au circuit pour que tout ne soit pas terne et fade. Les couleurs choisies respectent la convention des couleurs selon le type de composant du livre *Enginering Circuit Analysis*.

La documentation de MxGraph[8] dispose d'un tutoriel et de nombreux exemples simples ou complexes. Chaque exemple étant un prétexte pour mettre en évidence

l'usage d'une fonctionnalité ou la réalisation d'un petit projet. La liste de ces exemples peut se retrouver au lien cité en annexe [9]. Sur base de ces exemples, la création et la modification des fils ont été implémentées.

Les fils (arêtes du graphe) sont rectilignes et forment exclusivement des angles droits entre les composants. Les fils formant le circuit en tant que tel sont noirs, là où les fils connectés aux labels de tensions sont de couleur grise pour mettre en exergue le fait qu'ils n'ont pas d'influence sur les tensions et courants du circuit.

MxGraph est utilisé à trois endroits différents de l'application et pour chaque endroit, son utilisation est différente : le graphe éditable, le graphe de l'exercice et le graphe d'une ancienne soumission. Le code commun à ces trois versions se trouve dans une même fonction et d'autres fonctions sont présentes pour être utilisées par une ou plusieurs des trois versions du graphe.

3.3.1 MxGraph éditable

Un fichier HTML définit le graphe éditable pour créer le circuit électrique ou modifier ses attributs. Le graphe est d'abord récupéré grâce à la librairie jQuery. En effet, le graphe est stocké sous forme de XML dans un élément HTML invisible aux yeux de l'utilisateur. MxGraph permet d'encoder en format XML tous les composants, leurs attributs et les fils qui les relient. Ce XML est enregistré et transformé de telle sorte qu'il soit utilisable pour la simulation (cfr section NGSpice). Pour ce graphe, les composants et les fils doivent pouvoir être déplacés, modifiés et supprimés. Ces fonctionnalités sont définies grâce aux constantes de mxGraph[10].

Une fonction définit les panneaux de configuration de chaque élément. En effet, le nombre et la nature des attributs varient selon le type du composant. Cette fonction permet de s'assurer de modifier uniquement et exhaustivement les attributs qui ont du sens en fonction du composant.

Ensuite, le graphe peut être soit modifié manuellement soit généré aléatoirement. Les modifications manuelles sont traitées grâce à la boite à outils de mxGraph. Pour chaque élément de la boite à outils, une image et un composant sont donnés. Lorsque l'élément est sélectionné et posé, les attributs de l'élément sont spécifiés (nom, couleur, valeurs, ...). En ce qui concerne la génération aléatoire, une requête Ajax est envoyée au serveur pour créer un nouveau circuit avec les paramètres (nombre de cycles et type d'exercice) passés en argument.

Finalement, à chaque modification du graphe, le **XML** est mis à jour dans l'élément HTML pour être enregistré une fois terminé dans le fichier task.yaml.

3.3.2 MxGraph pour l'étudiant

Un autre fichier HTML définit le graphe de l'exercice utilisé par les étudiants. Le graphe peut, soit être récupéré dans un XML passé en argument, soit créé

aléatoirement. Le XML est simplement décodé grâce aux fonctions de mxGraph. Le graphe est généré aléatoirement si l'utilisateur active la fonctionnalité *Random* d'INGInious. Comme précédemment, une requête Ajax est alors envoyée au serveur. Cette fois-ci, les composants et les fils ne doivent pas pouvoir être modifiés pour éviter que l'étudiant perturbe le bon fonctionnement de l'application. Ces paramètres sont définis dans une fonction définissant les constantes de mxGraph. Une fonction définit les panneaux de configuration de chaque élément : encoder la réponse ou ajouter un nom de courant. Le nouveau nom de courant est ajouté au label du composant, à la suite de son nom.

Deux boutons sont aussi présents, l'ajout d'un nom de tension ou d'une équation. L'ajout d'un nouveau nom de noeud sur un fil crée en fait un nouveau composant avec comme label, le nom de la tension. Ce composant a un seul port et est connecté au composant source du fil par un fil invisible pour des questions d'esthétique. Le nouveau nom doit être différent des noms des autres labels qui sont donc enregistrés dans une liste. L'ajout d'une équation crée simplement un objet texte input qui sera par la suite interprété.

Lorsque l'étudiant soumet sa solution, ses équations et le xml du graphe sont envoyés dans les inputs (equations-submit et value-submit) pour ensuite être analysés. Les équations sont encodées dans l'élément input de la manière suivante :

id **ID** equation **NextEquation** id **ID** equation

Le tag NextEquation permet de séparer chaque équation alors que le tag ID permet de séparer l'équation de son id.

3.3.3 MxGraph ancienne soumission

Ce graphe se rapproche fortement du graphe de l'exercice, mais possède quelques variations. Tout d'abord, avant de charger l'ancienne soumission, le circuit actuel et les boutons doivent être enlevés ainsi que toutes les équations s'il y en avait. Ensuite, le XML et les équations de l'ancienne soumission sont récupérés pour être affichés. Le XML et les équations sont décodés respectivement grâce à la fonction de mxGraph et aux tags ID et NextEquation.

3.4 Conclusion

Dans cette section, toutes les fonctionnalités de l'interface ont été détaillées. La partie **Résolution** concerne principalement les étudiants et essaye de leur proposer le plus de liberté possible pour comprendre en profondeur les exercices. La partie **Création** s'intéresse aux professeurs, elle permet de créer tous les circuits possibles

avec les composants disponibles. Finalement, l'implémentation de l'interface a été réalisée grâce à la librairie mxGraph.

Chapitre 4

Simulation

La simulation des circuits interactifs permet premièrement de récupérer la valeur de la solution, mais aussi toutes les valeurs des mesures de courants ou de tensions de l'étudiant. Ceci permet de vérifier la solution finale et les équations.

4.1 NGSpice

NGSpice est un simulateur open source spice pour circuits électroniques. Ce simulateur n'utilise pas d'interface graphique permettant d'utiliser des circuits de manières schématiques, mais il a l'avantage de pouvoir prendre en entrée, des lignes de commande ou un fichier. Ces fichiers contiennent principalement trois éléments :

1. Une netlist qui est la représentation en code du circuit électrique.
2. Une commande de simulation qui va résoudre et calculer les différentes valeurs du circuit suivant les caractéristiques de la commande (analyse en courant continu, analyse en courant alternatif, etc).
3. Une liste de mesures. C'est la liste des valeurs désirées par l'utilisateur et cela conditionne dès lors les valeurs de retour de Spice.

La librairie MxGraph utilisée pour afficher nos circuits dans INGInious stocke la représentation sous forme d'un fichier **XML**. Afin de simuler le circuit et de vérifier la réponse de l'étudiant, il faut désormais récupérer la représentation **XML** de notre circuit et la convertir en Netlist Spice équivalente. Une fois la netlist obtenue, il reste à résoudre le circuit avec la commande de simulation adéquate. En effet, Spice propose de nombreuses manières de simuler le circuit. Néanmoins, il faut une commande convenant aux exercices proposés. La réponse de l'étudiant est alors comparée avec les résultats de la simulation. Dans le cadre des exercices, la valeur

pertinente à mesurer est évidemment la question de l'exercice (la tension entre deux noeuds, le courant passant dans un composant). De plus, d'autres valeurs peuvent également être pertinentes notamment dans la génération de feedback adéquat.

Premièrement, il faut créer le fichier de simulation pris en entrée par NGSpice. L'objectif est donc de former les 3 éléments, vu plus haut, qui forment un fichier Spice. Dans un premier temps, il faut convertir le fichier **xml** en netlist Spice. Ensuite, inscrire la commande de simulation à la suite de la netlist. Et finalement déterminer les mesures pertinentes afin de prodiguer assez de données pour qu'il soit possible d'en tirer à la fois une correction et un feedback utile. Le fichier ainsi formé peut être transmis à NGSpice . De plus, ce programme collecte également les réponses de l'étudiant et les stocke dans un fichier à part.

A ce stade, NGSpice exécute le fichier Spice. La sortie du programme retourne les résultats de la simulation. Ces résultats, à l'instar des réponses de l'étudiant, sont récupérés et stockés

Une fois ces données récupérées, il suffit de comparer les solutions de l'étudiant et les valeurs obtenues suite à la simulation. Après comparaison, il reste à déterminer si l'étudiant a réussi ou échoué la tâche INGInious. Le fonctionnement global est représenté sous la figure 4.1

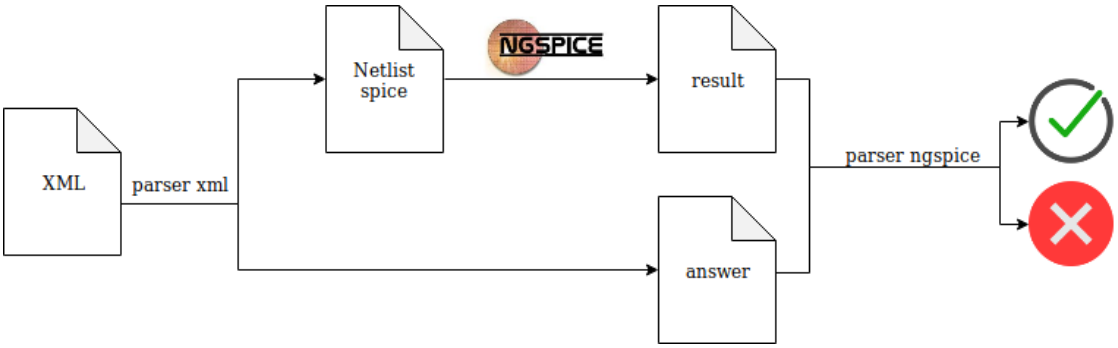


Figure 4.1: Vérification d'un circuit à l'aide de NGSpice

4.1.1 Structure des circuits en XML et en Netlist

Détaillons tout d'abord comment un circuit quelconque est représenté sous forme **XML** (figure 4.2). Tout d'abord, il y a des balises générées par la librairie qui assure son bon fonctionnement. Chaque composant est représenté par une balise <Component>. Cette balise contient les attributs principaux du composant :

- Identifiant : un id pour identifier le composant. C'est par cet id que le composant est relié aux autres composants.
- Type : source de tension/courant, résistance, inductance, etc.

- Nom : le nom du composant. Le nom est généré suivant le type du composant (une lettre suivant le type) et le nombre de composants du même type déjà présent (par exemple $R2$ lors de l'ajout de la deuxième résistance).

- La valeur : la valeur du composant suivant le type (Ohm [Ω], Volt [V], Ampère [A]).

Dans chaque balise <Component>, une balise <MxCell> et <MxGeometry> servent au rendu graphique. Les informations contenues dans ces balises sont donc inutiles dans la transformation en netlist Spice. La masse est indiquée pour sa part par la balise <Ground>. Celle-ci n'a pas de caractéristique spécifique, il a donc qu'un seul attribut id (il dispose malgré tout des balises <MxCell> et <MxGeometry> ayant la même utilité que pour les autres composants).

Les fils sont eux représentés par des balises <MxCell> également. Celles-ci sont au même niveau dans l'arborescence que les balises <Components> du **XML**. Il n'y a donc pas d'ambiguïté entre les balises <MxCell> qui servent au rendu des composants et celles qui représentent des fils. Lorsqu'une balise <MxCell> représente un fil, il a comme attribut :

- Identifiant : un id pour identifier le fil.

- Source : l'id du composant à partir duquel le fil "part".

- Target : l'id du composant où le fil "arrive".

- Edge : un booléen qui indique que l'élément est un "edge".

- Style : un attribut qui sert au rendu graphique, mais également qui indique les ports à partir duquel le fil part et arrive. Élément qui est essentiel dans la formation de la netlist.

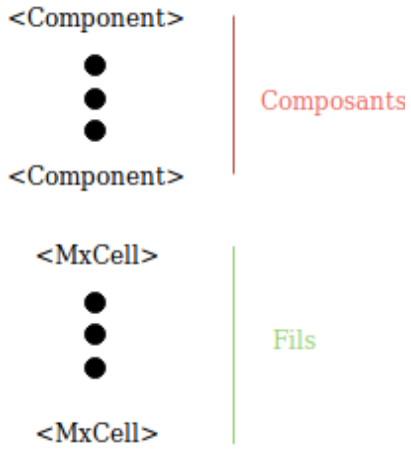


Figure 4.2: Format du xml mxgraph

Premièrement, les attributs source et target sont interchangeables. En effet, MxGraph génère l'arête en lui donnant une direction. Un composant départ (source) est connecté vers un composant destination (target). Néanmoins, dans un circuit (et de manière équivalente dans une netlist), un fil électrique ne possède pas de direction. L'important, c'est que les composants soient reliés sans qu'aucune considération de sens entre en compte. L'attribut style par contre à une grande importance. En effet connecter un composant à la borne négative ou la borne positive d'une source de tension sont deux possibilités radicalement différentes. De même pour une résistance ou tout autre composant, la borne choisie impacte la topologie du circuit comme montré sur la Figure 4.3.

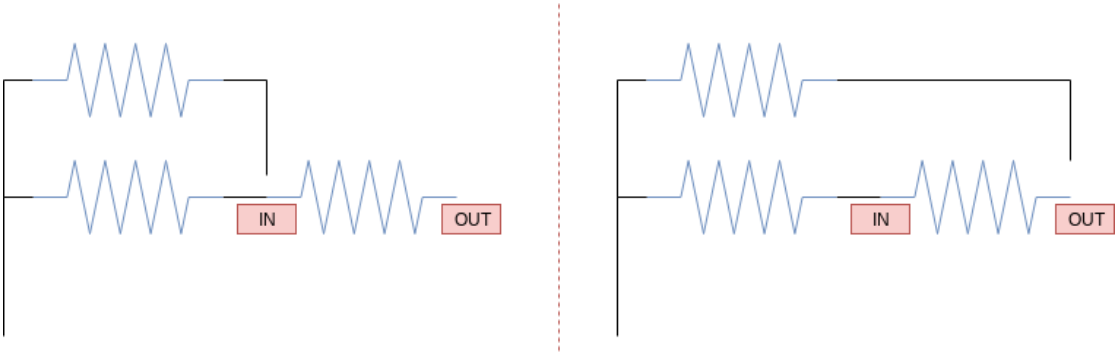


Figure 4.3: Impact du choix des bornes, à gauche la borne in de la résistance est choisie, à droite la borne out

L'information de la borne connectée est dès lors cruciale, car elle influe sur la netlist.

De plus, dans le format **XML**, un noeud du circuit peut être formé de plusieurs manières possibles. La Figure 4.4 montre un exemple. À gauche, le noeud est formé en connectant les deux résistances à la source tandis qu'à droite les deux résistances sont connectées et ensuite l'une des deux est finalement reliée à la source. Le résultat, d'un point de vue du circuit, est identique, peu importe la manière de procéder.

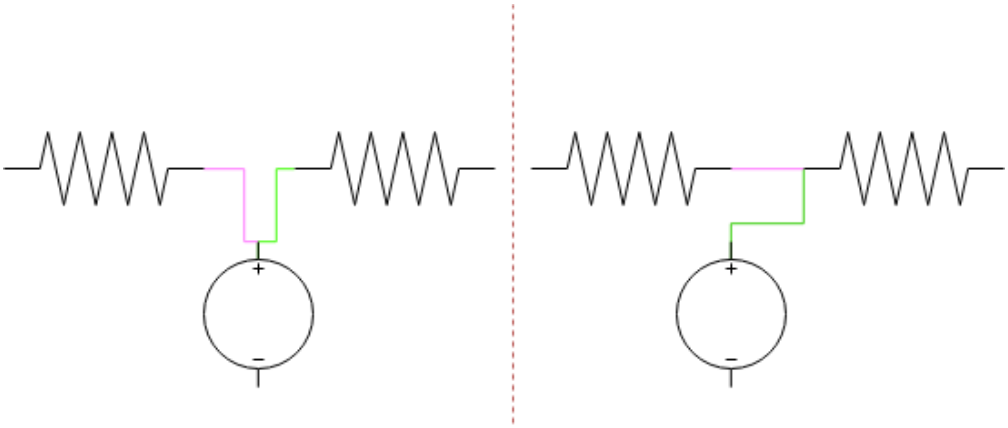


Figure 4.4: Un noeud peut être formé de plusieurs façons possibles

En résumé, le circuit est représenté sous forme **XML** par une liste de composants avec leurs différents attributs et une liste d'arêtes (une liste de fils) qui connectent un élément à un autre.

Une Netlist décrit un circuit de manière différente. À la place de relier un à un les composants, chacun dispose d'une ou plusieurs connexions à un noeud. Deux composants ayant une même connexion signifie qu'ils sont reliés directement par un fil. Ceci peut être vu comme un graphe où les arêtes sont les composants et les sommets les noeuds du circuit. Une netlist ne contient donc qu'une liste de composant avec, pour chacun, les connexions associées et les différents paramètres intrinsèques de ce composant. Les composants gérés actuellement ne disposent que d'une ou deux seules connexions. Une autre différence avec la représentation en **XML** concerne la masse. La masse n'est pas directement indiquée dans une netlist comme un élément constitutif du circuit (comme en **XML**, où elle est représentée et reliée aux composants). En Spice, on indique qu'un élément est connecté à la masse en nommant une de ses connexions par "0". En effet, alors que les autres connexions peuvent être nommées de n'importe quelle manière, le "0" est réservé pour indiquer le noeud de référence du circuit où la tension est nulle (autrement dit la masse).

4.1.2 Composants implémentés

`parser.xml` est capable actuellement de gérer 9 composants "constitutifs" différents:

- Source de courant et de tension
- Résistance
- Capacité
- Inductance
- Source de courant commandée en courant ou en tension
- Source de tension commandée en courant ou en tension

Ces composants ont tous deux ports et sont donc connectés à deux noeuds. Ces neuf composants sont appelés "constitutifs" car ils forment le circuit.

Le parser gère également les trois composants "informatifs" :

- Label de tension
- Label de courant
- Label de noeud

Ces composants n'influent pas directement sur le circuit. Ces labels servent à indiquer ce qu'il faut mesurer lors de la simulation du circuit. Mais ils jouent également un rôle essentiel d'un point de vue pédagogique. C'est par ces labels de tension ou du courant que l'étudiant pourra inscrire sa réponse (pour une tension demandée, il inscrira la tension qu'il a calculée dans le label de tension correspondant). De son côté, le label de noeud a un but d'apprentissage. Ce composant permet à l'étudiant de nommer des noeuds et à partir de ça, écrire et vérifier ses équations. Le programme va donc utiliser ces composants, non pas pour modifier la topologie du circuit, mais pour faire des mesures sur les noeuds demandés et inscrire les résultats de l'étudiant. Les labels de tension et de courant peuvent servir de question comme dit ci-dessus, peuvent indiquer une mesure et troisième utilité peuvent donner un nom à une différence de potentiel ou à un courant dans un fil. Dans ce cas-ci, ces labels sont utilisés avec des sources commandées pour indiquer leur dépendance. Cette dépendance est alors utile dans la description de la Netlist. En effet, le label devient alors un attribut de la source dépendante.

La simulation d'ampèremètre par le label de courant est en réalité une source de tension nulle. La raison est qu'ainsi le courant passant dans la source peut être mesuré sans impacter le circuit. C'est essentiel, car NGSpice n'offre pas la

possibilité de mesurer le courant dans un fil donné, il est dès lors obligatoire de passer par cette méthode pour mesurer un courant.

Il y a un dernier type de composant : l'impédance cachée. Elle a à la fois un but constitutif et un but informatif. En effet, ce composant est utilisé dans les exercices de type indirect. Une résolution directe consiste à donner un circuit et de calculer une mesure dans ce circuit (une différence de potentiel ou courant dans un composant/fil). Une résolution indirecte est donc un exercice inversé où on donne une mesure et où l'étudiant doit calculer la valeur d'une impédance. Dans ce cas, l'impédance fait partie du circuit, il est donc bel et bien constitutif, mais il est également informatif, car il contient la réponse de l'étudiant.

4.1.3 Formation des noeuds à partir du XML

Nous allons maintenant détailler comment passer d'une structure **XML** à une structure en graphe. La structure en graphe est un ensemble de noeuds et un ensemble d'arêtes qui relie ces noeuds. Dans le graphe utilisé, chaque arête est associée à un objet composant. Une fois le graphe créé à partir du **XML**, il suffit de parcourir les arêtes de ce graphe et d'y inscrire les composants associés pour ainsi former la netlist.

Globalement, il faut reformer les noeuds et trouver les composants associés. Une fois les noeuds obtenus et leurs composants trouvés, les différents noeuds sont reliés entre-eux par ces mêmes composants. À ce stade, les connexions de tous les composants sont dès lors connues et la netlist prête à être écrite.

Pour réaliser cela, le parser va, dans un premier temps, parcourir les éléments <MxCell> et former une liste de dictionnaire Python. Un dictionnaire pour chaque edge et qui contient la liste d'attribut suivant :

- Source : id de la source
- Target : id de la target
- Sourceport : port de la source
- Targetport : port de la target

Le parcours de cette liste va permettre de connaître le nombre de noeuds qui constituent notre graphe. Bien sûr, connaître le nombre de noeuds n'a qu'un intérêt limité. Petit à petit, la liste des noeuds que contient notre circuit est formée. Pour chaque noeud, toutes les arêtes (et évidemment le composant associé) liées à celui-ci sont stockées. Chaque noeud du graphe est alors également associé à une liste de tuple. Les tuples contenant l'id du composant connecté à ce noeud et le port à partir duquel il y est relié. Lorsqu'on parcourt la liste des arêtes venant du XML, trois cas possibles peuvent se présenter dans la mise à jour des noeuds :

- Dans le premier cas : les deux composants connectés par le fil ne disposent pas encore de connexion en commun à partir des ports où le fil est attaché. Dans cette situation, un nouveau noeud est tout simplement créé avec les deux composants et leur port respectif à partir duquel le fil "part" ou "arrive". Ce nouveau noeud est rajouté au graphe.

Figure 4.5: Création d'un noeud à partir d'une arête liant deux composants isolés

- Deuxième cas : l'arête relie un composant qui est déjà lié à un noeud et à un autre composant qui n'est pour l'instant connecté à rien (par le port où le fil est connecté). Dans ce cas-ci, le noeud est modifié et contient désormais le nouvel élément qui n'était relié à rien comme montré sur la figure 4.6. Un composant relié à un autre situé dans un noeud revient tout simplement à ce que ce premier composant appartienne à ce noeud. Le noeud possède désormais une nouvelle "arête". Le degré du noeud du graphe augmente alors de 1.

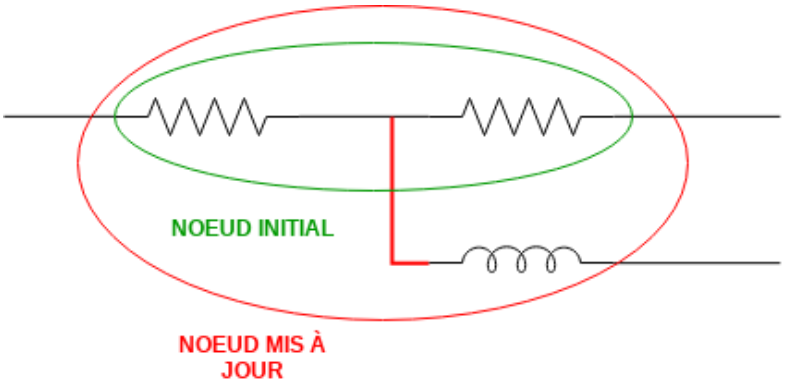


Figure 4.6: Mise à jour d'un noeud en rajoutant une arête

- Troisième et dernier cas : l'arête relie deux composants qui se trouvent déjà tous les deux dans un noeud (toujours par le port où le fil est connecté). Tous les éléments du premier et du deuxième noeud vont être rassemblés en un seul noeud. Ainsi, les deux noeuds seront fusionnés en un seul noeud, figure 4.7, le noeud ainsi formé est finalement ajouté au graphe. Les deux noeuds initiaux sont quant à eux supprimés du graphe.

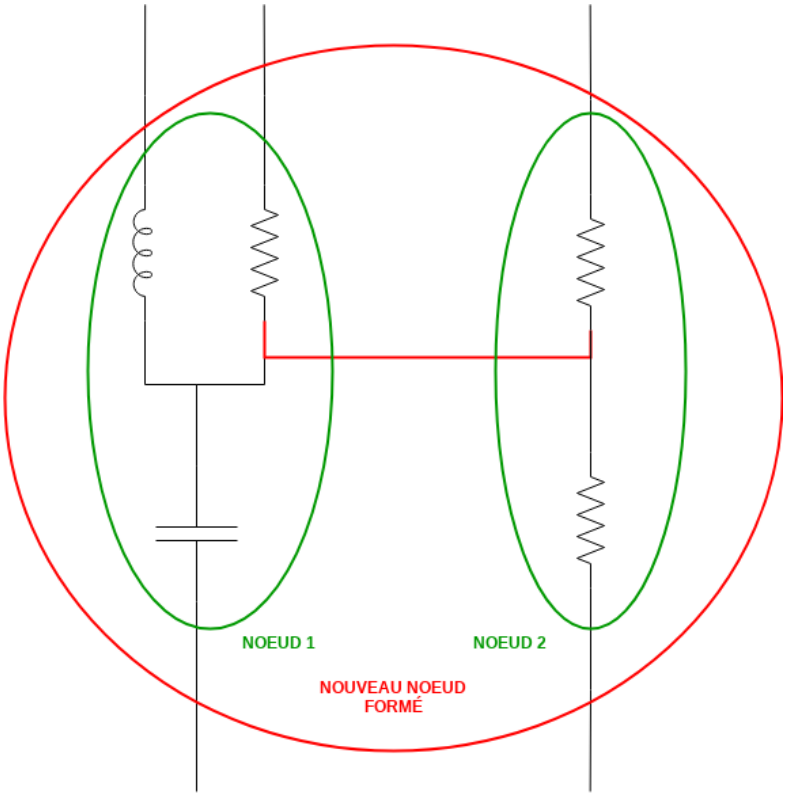


Figure 4.7: Fusion de deux noeuds

4.1.4 Formation de la Netlist

Une fois toutes les arêtes du **XML** parcourues, tous les noeuds de la netlist sont formés. À ce moment, le nombre de noeuds final du graphe est connu, mais le graphe ne contient encore réellement aucune arête et ne possède actuellement qu'un ensemble de noeuds. Bien que tous les composants reliés à ces noeuds sont connus, il suffit d'ajouter une arête entre deux noeuds qui ont un composant en commun. Normalement tous les composants du circuit sont reliés à deux noeuds (car pour rappel tous les composants implémentés n'ont que deux ports excepté la masse). Et ça concorde avec la structure en graphe utilisée, en effet une arête relie au maximum deux noeuds (si on exclut une possible extension de graphe vers hypergraphe). Si ce n'est pas le cas et qu'un composant est associé à au moins trois noeuds, cela signifie que le fichier **XML** représente un circuit erroné ou alors un circuit avec un composant non implémenté par exemple un transistor (ayant trois ports, il est forcément lié à trois noeuds). La netlist sera donc également invalide et sera impossible à simuler. Après avoir ajouté toutes les arêtes (une arête par composant), la construction du graphe est finalement terminée.

Une fois tous ces noeuds acquis, la netlist peut être écrite. Les balises <Component> du **XML** sont parcourues afin de créer un dictionnaire pour chaque élément. Ce dictionnaire contiendra toutes les valeurs essentielles de l'élément contenu dans la représentation **XML** (valeur, nom, type, etc). Les arêtes associées aux composants permettent de savoir facilement quels noeuds sont liés via ces mêmes composants. Le nom des noeuds est lié à leur index dans la liste en sachant que le noeud de référence, le noeud relié à la masse, a obligatoirement le numéro 0.

Ces dictionnaires formés, il suffit de parcourir les arêtes du graphe et d'inscrire pour chaque arête la ligne correspondant au composant associé en netlist Spice. Le format NGSpice des différents éléments implémentés est détaillé ci-dessous. À savoir que l'outil NGSpice permet pour les différents composants de choisir des paramètres supplémentaires et optionnels dans le but de simulation bien précise. Ici, uniquement les paramètres obligatoires et les options pertinentes à notre simulation sont explicités. Les différentes options et paramètres possibles peuvent être trouvés plus en détail dans le manuel NGSpice [11].

Pour une résistance, une inductance ou une capacité, le format est semblable et est le suivant :

```
RXXXXXXXX N+ N- value
CXXXXXXXX N+ N- value
LXXXXXXXX N+ N- value
```

Tout d'abord, le nom du composant est indiqué. Ce nom en Spice déterminera le type du composant. Le nom commence par R, C ou L pour respectivement une résistance, une capacité ou une inductance. On peut directement utiliser le nom trouvé dans le fichier **XML**, car cette logique y est déjà respectée. `N+` et `N-` sont les deux noeuds liés au composant. Ces trois composants n'ayant pas réellement de "sens", l'inversion de `N+` et `N-` n'a guère d'incidence dans le comportement du circuit. Néanmoins, pour garder une cohérence, une convention a été choisie et `N+` est le noeud lié au port *in* alors que `N-` est lié à *out*. `value` est comme son nom l'indique la valeur du composant, en Ohm pour les résistances. Les valeurs des inductances et des capacités sont particulières et seront discutées dans la partie simulation.

Le format d'une source de tension et d'une source de courant :

```
VXXXXXXX N+ N- <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
IXXXXXXX N+ N- <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
```

Comme précédemment, le composant est identifié par son nom. Tout élément dont le nom commence par V ou I est donc respectivement une source de tension ou une source de courant. Le nom donné dans le **XML** peut une nouvelle fois être

utilisé pour la même raison que précédemment. `N+` et `N-` sont toujours les deux noeuds. Cependant, cette fois-ci, ils ne peuvent pas être inversés. En effet, une source de tension et une source de courant ont un sens. Pour la source de tension, il faut que `N+` soit le noeud lié au port *N* (Nord) et `N-` lié au port *S* (Sud). Dans le cas de source de courant, `N+ (N-)` est lié au port *Sud* (respectivement *Nord*).

L'une des difficultés mise en évidence ici est que tous les stencils utilisés n'ont pas la même logique pour leurs ports. Pour l'une des sources, `N+` est lié au port Nord tandis que c'est le port Sud pour l'autre. Une solution possible aurait été de réécrire les stencils utilisés afin qu'ils aient tous les mêmes noms de ports et que les noms soient assignés à `N+/N-` de manière uniformisée (tous les ports Nord (Sud) lié à `N+ (N-)` par exemple) . Ainsi, tous les composants seraient gérés de la même façon. La solution choisie est de laisser les stencils tel qu'ils sont, mais de vérifier le type de composant afin d'attribuer les noeuds, suivant la logique établie par NGSpice.

`DC/TRAN VALUE` est la valeur continue et transitoire de la source. Dans le cas de simulation en courant alternatif, cette valeur vaut 0.

`ACMAG` est la magnitude AC (courant alternatif) et `ACPHASE` la phase AC. Pour évaluer le circuit, la source est paramétrée à ces valeurs. Plus de détails seront fournis sur la simulation et les mesures par la suite. `ACMAG` et `ACPHASE` sont des valeurs issues du **XML** et donc normalement présentes pour toutes sources dans le circuit.

Le format pour une source commandée en tension est le suivant :

```
GXXXXXXX N+ N- NC+ NC- VALUE
EXXXXXXX N+ N- NC+ NC- VALUE
```

Le type de composant est de nouveau déterminé par le nom. Le nom commence par G (E) pour une source de courant (respectivement tension) commandée en tension. Dans ce cas-ci, le nom issu du **XML** ne peut plus être utilisé. En effet dans l'éditeur de circuit, lorsqu'un utilisateur ajoute une source commandée, il est impossible de savoir à l'avance si la source sera commandée en tension ou en courant. Ce choix est fait par l'utilisateur lorsqu'il crée ou édite un circuit et dépend du label associé à cette source. Or, en netlist NGSpice le nom est différent selon que la source est commandée en tension ou en courant. Le moyen utilisé dans le parser est de vérifier le type de la source (tension/courant) puis de vérifier le type de dépendance assigné via un label de tension ou de courant. Suivant cela, le nom venant du **XML** est pris (cela empêche les doublons, car les noms générés par l'éditeur sont uniques) et on y concatène au début la lettre qui convient. Le nom du composant aurait pu être édité du côté **XML** lorsque la dépendance est choisie, mais le choix a été de faire cela du côté du parser, car il était aisé de retrouver les différents types (sources et label).

Le comportement de `N+` et `N-` est identique au cas des sources indépendantes. Suivant le type de port, un des noeuds y est associé.

`NC+` et `NC-` sont les deux noeuds qui représentent la différence de potentiel. `NC+` et `NC-` sont respectivement les noeuds de contrôle positif et négatif. Dans le cas d'une source commandée, un label a été attribué. Pour rappel, le label est seulement un élément qui indique certaines informations, mais qui n'impacte pas le circuit. Ici, les informations données sont les deux noeuds qui représentent ensemble la tension de commande. Une fois le label récupéré, `NC+ (NC-)` est le noeud associé au port N (respectivement S) du label.

`VALUE` est le facteur de la source en [V/V] ([A/V]) pour une source de tension (respectivement source de courant)) commandée en tension.

Finalement, le format pour une source commandée en courant est le suivant :

```
FXXXXXXX N+ N- VNAME VALUE
HXXXXXXX N+ N- VNAME VALUE
```

Comme toujours le nom désigne le type de composant. Le nom commence par F (H) pour une source de courant (respectivement tension) commandée en courant. Les sources commandées en courant se comportent et se manient de la même façon que les sources commandées en tension. La seule différence notable est le champ `VNAME` à la place de `NC+` et `NC-`. `VNAME` est le nom de la source de tension dans laquelle le courant de contrôle circule. Une source courant-dépendante ayant forcément un label de courant associé, il suffit de remplacer `VNAME` par le nom du label de courant (qui est, pour rappel, une source de tension nulle).

4.1.5 Simulations

Tous les composants sont inscrits, la Netlist est formée, mais il reste à simuler le circuit. Sur base de ces simulations, les valeurs pertinentes seront mesurées et finalement ces mesures seront comparées avec la réponse de l'étudiant et un feedback sera produit selon le résultat.

Les exercices proposés sont des cas classiques de circuits AC en état stable comme on peut le retrouver dans le cadre du cours LELEC1370 : Circuits et Mesures. La commande `.ac` de NGSpice est utilisée pour effectuer une analyse AC de ces circuits. Une analyse AC [11] interprète la solution sinusoïdale d'un système décrit à un état stable et à une fréquence particulière (un ensemble de fréquences est également possible, mais n'a guère d'intérêt ici). Pour réaliser cette analyse AC, NGSpice calcule d'abord le point de fonctionnement en continu du circuit (point DC) et détermine la linéarisation pour tous les dispositifs non linéaires du circuit. Le circuit linéaire résultant est ensuite analysé sur une plage de fréquences spécifiée par l'utilisateur. La commande est décrite comme suivant :

```
.ac lin np fstart fstop
```

`lin`, `np`, `fstart` et `fstop` sont quatre paramètres qui servent à établir une plage de fréquence d'analyse du circuit. Le premier paramètre définit le type de variation (linéaire, logarithmique), le deuxième le nombre de points lors de cette variation, le troisième et quatrième paramètres sont respectivement la fréquence de départ et la fréquence de fin. Dans notre cas, le circuit est analysé à fréquence unique. Dès lors, peu importe le type de variation choisie. La valeur de `np` est donc de 1 et les deux fréquences qui définissent les limites sont donc les mêmes.

Mais quelle fréquence choisir? Dans les exercices proposés sur INGInious, la fréquence des sources n'est jamais mentionnée. Pour rappel, on peut pour chaque impédance Z écrire :

$$V = Z \cdot I \tag{4.1}$$

Cette valeur Z varie selon la fréquence. En effet :

- **Résistance** : $Z = R$ où R est la grandeur en [Ω] de la résistance. L'impédance d'une résistance est purement réelle. Le composant ne crée aucun décalage entre courant et tension.

- **Inductance** : $Z = j \cdot \omega \cdot L$ où L est la grandeur en Henri de l'inductance et où ω est la pulsation. L'impédance d'une inductance est purement imaginaire. Son amplitude varie linéairement avec la fréquence. À très basse fréquence, l'inductance est assimilée à un fil tandis qu'à très haute fréquence elle est assimilée à un circuit ouvert.

- **Capacité** : $Z = \frac{1}{j \cdot \omega \cdot C}$ où C est la grandeur en Farad [F] de la capacité. Comme l'inductance, l'impédance d'une capacité est purement imaginaire. Son amplitude est inversement proportionnelle à la fréquence. Son comportement est donc inverse par rapport à l'inductance : la capacité peut être considérée comme un circuit ouvert à très basse fréquence et comme un fil à très haute fréquence.

Cependant, dans les exercices, la valeur des impédances est directement indiquée: pour une résistance, la valeur est un nombre réel et un nombre purement imaginaire pour les capacités et les inductances. Les valeurs des impédances étant déjà données, l'étudiant n'a donc pas à s'inquiéter de la valeur de fréquence des sources.

Dans le cas de la résistance, ça n'a pas beaucoup d'impact, Z étant égal à R. La mesure en Ω donne directement la bonne impédance. Pour les deux autres composants, la valeur récupérée est la valeur absolue de la partie imaginaire de l'impédance équivalente. Autrement dit, l'impédance équivalente étant un nombre imaginaire pur de la forme ai, $a \in Z$, $|a|$ est récupéré. Mais malheureusement

ce n'est pas directement les mesures L et C alors que dans la netlist, c'est bel et bien les unités internationales qui sont utilisées (Farad pour une capacité, Henri pour une inductance). Pour régler ce souci, on va premièrement fixer pour une inductance la valeur de l'impédance (soit a) et l'inverse pour une capacité (soit $\frac{1}{a}$). Étant donné que, comme vu plus haut, l'impédance est proportionnelle à L et inversement proportionnelle à C. Ensuite, ω est fixé à 1. Sachant que :

$$\omega = 2 \cdot \pi \cdot f \tag{4.2}$$

La fréquence idéale pour simuler est donc :

$$f = \frac{1}{2 \cdot \pi} \tag{4.3}$$

Dans la commande NGSpice, une valeur approximée est utilisée $\frac{1}{2 \cdot \pi} \approx 0.1591$. La commande de simulation des circuits revient finalement à :

```
.ac lin 1 0.1591 0.1591
```

De cette manière lors de la simulation, les résistances, inductances et capacités ont la même impédance que dans les exercices.

Une autre solution possible aurait été de mettre directement la mesure des capacités/inductances et de mentionner la fréquence des sources dans le circuit. Dans le fichier NGSpice, il aurait suffi alors de remplacer directement les mesures dans la netlist et dans la commande remplacer `fstart` et `fstop` par la fréquence donnée. La solution précédente a été choisie, car la fréquence n'a finalement que peu d'intérêt dans la résolution de ce type d'exercice. En effet, la fréquence n'est utilisée que dans les formules afin d'obtenir les impédances des inductances/capacités. Pensant que l'intérêt des exercices ne réside pas dans l'application de formule, mais plus dans la résolution du circuit, la fréquence n'est pas mentionnée. C'est le programme qui convertit les valeurs de telle sorte que le circuit se comporte comme dans l'exercice. Ce choix a été fait suivant la catégorie spécifique des exercices mis en place. Il est évident que si de nouveaux types d'exercices sont implémentés à l'avenir par exemple : un dimensionnement de filtre, la fréquence aura alors un rôle important à jouer. Le choix de masquer la fréquence est donc bien lié au type d'exercice, actuellement implémenté dans l'extension et non une règle absolue dans le cadre d'analyse de circuits AC.

Maintenant que la netlist est formée et que la commande de simulation est prête, il reste à mesurer et à comparer les résultats de l'étudiant.

4.1.6 Mesures

À la suite de la netlist et de la commande de simulation, diverses commandes supplémentaires peuvent être ajoutées afin de mesurer certains courants/tensions.

Pour rappel, les mesures se font via les composants informatifs, à savoir : les labels de tension ou de courant et les "nodes". Ces composants peuvent servir de "mesure" ou de "question". Ici, c'est bien le premier cas qui nous intéresse. Un étudiant peut également utiliser un courant passant dans un composant spécifique.

Grâce à la commande print et aux fonctions $V()$ et $I()$, les différentes mesures peuvent être inscrites dans un fichier et ainsi être stockées. Avec comme format : une ligne indiquant le nom du label mesuré, suivie juste en dessous d'une ligne contenant le résultat. Par exemple :

```
print of Label V0
v(1) - 0 = 5.484600e-01,2.049003e+00
```

Figure 4.8: Mesure du label V0, V0 étant la différence de tension entre le noeud 1 et la masse du circuit. Le résultat se présente en deux nombres, la partie réelle et la partie imaginaire de la mesure.

La fonction $V()$ prend comme paramètre le nom d'un noeud et retourne la valeur de tension de ce noeud par rapport à la masse. Pour un composant du type Label de tension, c'est la différence entre `N+` et `N-` qui est inscrit dans les mesures. Par contre, NGSpice n'accepte pas comme notation $V(0)$, "0" étant pour rappel le nom assigné d'office à la masse. En effet, ça n'a que peu d'intérêt de demander la tension de la masse sachant qu'elle vaut par définition 0. Cela oblige à vérifier le nom du noeud à chaque appel de la fonction $V()$ et dans le cas où c'est le noeud 0, noter 0 à la place de $V(0)$.

La fonction $I()$ de NGSpice est malheureusement uniquement capable de mesurer le courant aux bornes d'une source de tension [11]. Le courant passant dans une source de courant est trivial (valeur de la source), alors que les courants passant dans des impédances peuvent être obtenus grâce à la loi d'Ohm $V = Z \cdot I$. Puisque nous disposons de `N+` et `N-`, il suffit de calculer :

$$\frac{V(N+) - V(N-)}{Z} \tag{4.4}$$

Une autre solution pour mesurer un courant dans une branche est de mettre une source de tension en série et ainsi mesurer le courant passant dans cette source.

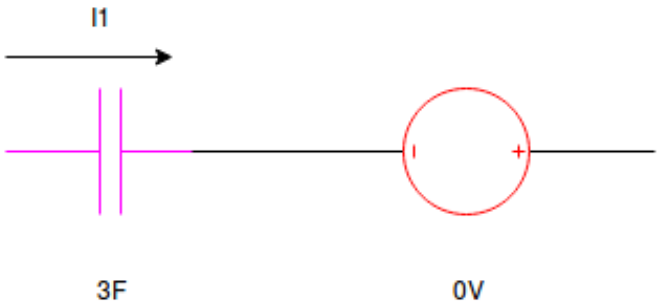


Figure 4.9: Mesure du courant passant dans la capacité à l'aide d'une source de tension

Sur la figure 4.9 par exemple, le courant passant dans la capacité peut être obtenu via la source de tension. La source de tension étant à 0 volt, sa présence n'influe en rien le comportement général du circuit. Cette solution n'a pas été retenue, car pour chaque courant mesuré, une source de tension doit être rajoutée dans la netlist. L'application de la loi d'Ohm semble bien plus appropriée et facile à mettre en place, car elle ne nécessite aucune modification de la netlist.

4.1.7 Topologie non acceptée par Spice et solutions

NGSpice est un outil très puissant, mais il y a certaines topologies de circuit que NGSpice échoue à calculer. Ces cas ne sont pas des failles du programme ou un quelconque dysfonctionnement, mais bien un choix de conception. Cela signifie que ces configurations sont connues [12] et qu'il suffit d'y porter attention pour ne pas causer une erreur lors de la simulation. Cela arrive notamment dans deux cas :

- Des capacités en série.
- Une boucle composée uniquement d'une source de tension et d'une inductance.

Les raisons pour lesquels Spice n'arrive pas à gérer ces situations proviennent de sa façon à réaliser une analyse DC et par extension une analyse AC (une analyse AC passe par une analyse DC [11] comme il est mentionné dans la partie simulation). Lors d'une telle analyse, Spice considère toutes les capacités comme des circuits ouverts et toutes les inductances comme des courts-circuits. De plus, les courts-circuits peuvent être assimilés à des impédances de $0[\Omega]$ là où au contraire les circuits ouverts sont semblables à une impédance de $\infty[\Omega]$. Ces impédances aux valeurs extrêmes entraînent des calculs qui font intervenir l'infini et autres valeurs difficilement manipulables par un ordinateur. Malheureusement cela provoque l'arrêt de l'analyse.

Heureusement, il existe des solutions pour que l'analyse se déroule correctement. Cela nécessite l'ajout de résistances aux endroits et aux valeurs appropriés et donc cela nécessite par conséquent une légère modification de la topologie des circuits. Ces modifications, si elles sont faites correctement, ne changent que de manière infinitésimale les mesures et comportements globaux des circuits. Elles sont donc dans certaines situations obligatoires, mais ne limitent en aucun cas les possibilités d'exercices. D'ailleurs ces modifications sont invisibles du point de vue de l'utilisateur et ne servent qu'au bon déroulement de la résolution.

Plus précisément, intéressons-nous au premier cas concernant les capacités en série. La simulation NGSpice échoue, car elle n'arrive pas à déterminer la valeur de tension entre les deux capacités. Comme dit plus haut, les capacités peuvent être considérées comme des circuits ouverts. Spice se voit dès lors incapable de déterminer la tension d'un noeud qui se trouve entre deux circuits ouverts. Ce noeud n'a guère de sens, car il ne peut pas exister s'il n'est relié à rien d'autres dans le circuit. C'est pourquoi après de nombreuses et vaines itérations, NGSpice échoue à trouver une solution. Pour remédier à ce problème, Spice a besoin d'un chemin DC à chaque capacité pour une analyse.

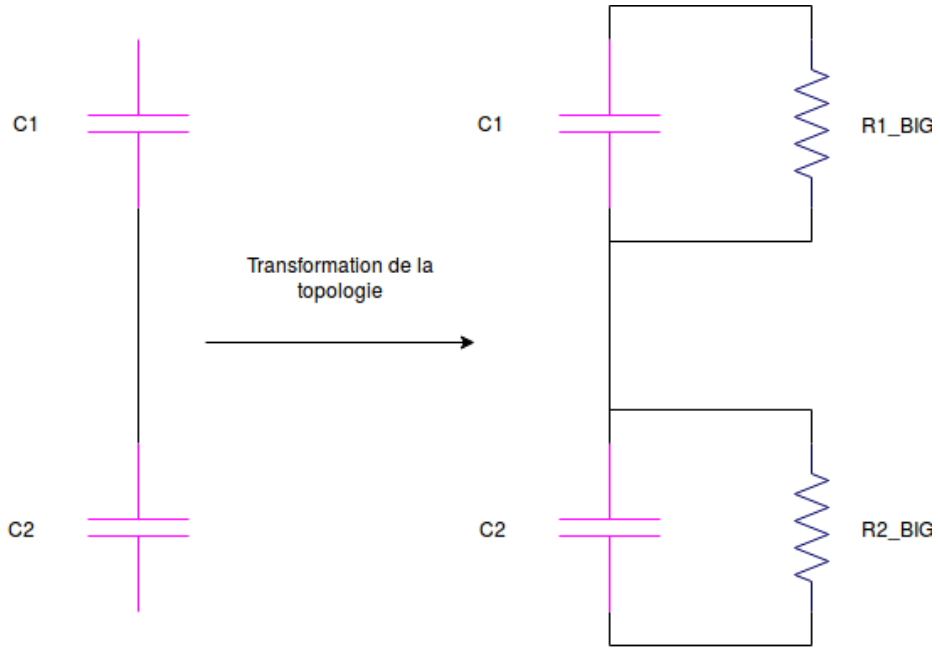


Figure 4.10: Transformation de la topologie de deux capacités en série par l'ajout de deux "grosses" résistances

La figure 4.10 montre la transformation effectuée pour gérer le cas de capacités en séries. La solution est de rajouter pour chaque capacité une résistance de grande

taille en parallèle. Ainsi, un chemin DC est créé et le noeud entre les deux capacités ne se retrouve plus isolé, sa tension est donc désormais calculable. La grandeur de résistance reste à déterminer avec pour objectif de limiter au maximum l'influence de ces composants. Ces deux résistances doivent donc être les plus grandes possible afin que le courant, passant dans ces résistances, soit négligeable et limiter ainsi les perturbations sur le circuit original. En pratique, des résistances de $10^{10}[\omega]$ ont été utilisées.

Une remarque : dans la situation montrée figure 4.10, une seule résistance en parallèle d'une capacité aurait suffit à régler le problème et permettre à NGSpice de calculer la solution. Le chemin DC était formé, le noeud entre les deux capacités n'était plus uniquement relié à des circuits ouverts et la simulation aurait donc pu aller à son terme. Il est donc important de souligner que de mettre systématiquement une résistance en parallèle à une capacité est suffisant pour résoudre le problème, mais pas forcément nécessaire. Néanmoins, l'ajout de ces résistances étant négligeable pour le circuit (même pour un nombre important de capacités), trouver le nombre minimum suivant toutes les topologies possibles semble inutile.

D'un point de vue de la netlist Spice et du graphe correspondant, quelles sont donc les modifications à apporter? Le nombre des noeuds du graphe reste inchangé après cette modification. Cependant, une arête est bien évidemment ajoutée pour chaque ajout de résistance. Dorénavant, lorsque deux noeuds sont reliés par une arête "Capacité", une arête "Résistance" supplémentaire les relie également. Lors de l'inscription de la netlist, il faut ajouter dans la netlist cette Résistance. Le nom de la résistance est généré à partir du nom de la capacité, ainsi si le nom de la capacité est $C1$ le nom de la résistance associée est simplement $RC1$. Concernant les noeuds, ils sont bien évidemment identiques pour la capacité et la résistance. Finalement, comme mentionné plus haut, la valeur choisie est de $10G[\Omega]$

Passons à la deuxième situation, à savoir une boucle composée uniquement d'une source et d'une inductance. Lors de l'analyse DC, l'inductance assimilée à un court-circuit entraîne un courant infini dans la boucle. Cette valeur est inadaptée pour Spice et empêche le bon déroulement de l'analyse. La figure 4.11 montre la transformation de configuration afin de borner le courant. Cette solution est toute simple et consiste à ajouter une résistance en série de l'inductance et ainsi empêcher que la branche soit un court-circuit. Néanmoins, cette résistance doit être la plus petite possible afin que le circuit transformé ait un comportement équivalent au circuit initial.

Du côté de la netlist l'ajout d'une résistance en série augmente le nombre de noeuds et l'agencement des arêtes est légèrement modifié. En effet, le noeud situé entre l'inductance et la résistance récemment rajoutée est évidemment créé et l'arête inductance relie l'un des noeuds de départ au nouveau, tandis que l'arête résistance relie le nouveau à l'autre noeud de départ de l'inductance.

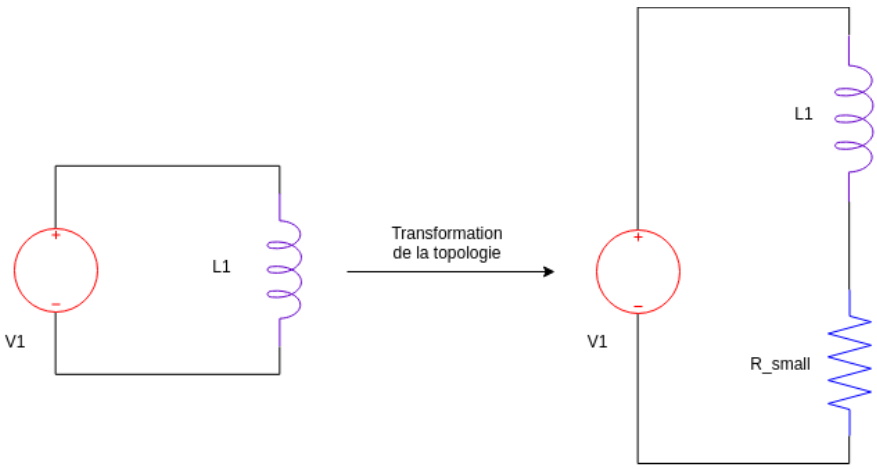


Figure 4.11: Transformation de la topologie d'une source en série avec une inductance par l'ajout d'une "petite" résistance

4.2 Correction

Certains labels servent également de questions. Dans ce cas-là en plus de mesurer la valeur du label, la valeur de l'étudiant est également inscrite dans un fichier. Dans la majorité des cas, un exercice demandera une valeur particulière de tension/courant. Néanmoins, des exercices demandant plusieurs valeurs sont tout à fait envisageables et possibles. Ainsi, un exercice en deux étapes avec une première question relativement facile qui découle sur une deuxième plus complexe est tout à fait réalisable.

Maintenant avec l'aide des deux fichiers générés contenant pour l'un les résultats de la simulation et pour l'autre les réponses de l'étudiant, la vérification des réponses peut être réalisée. Le fonctionnement de ce correcteur est relativement simple. Pour chaque réponse, le programme va chercher la valeur correspondante mesurée de la simulation et voir si la réponse donnée se situe dans une plage de données proches. Si ce n'est pas le cas, la réponse pour cette question est considérée comme fausse.

Maintenant se pose la question de quelle plage définir. Dans un premier temps, la plage acceptée est définie comme ceci :

$$[x - \frac{3 \cdot x}{100}, x + \frac{3 \cdot x}{100}] \tag{4.5}$$

Où x est la valeur mesurée de la simulation.

Ces 3 % de variation autour de la valeur permettent ainsi une légère marge d'erreur d'approximation à l'étudiant sans pour autant accepter des valeurs réellement éloignées. Il y a toujours malheureusement un risque d'accepter une réponse

obtenue par des calculs erronés, mais qui par chance tombe proche de la solution. À part choisir une plage plus réduite, il n'y a guère de solution au problème. Le souci étant qu'une plage plus réduite peut potentiellement refuser une réponse obtenue avec des calculs corrects, mais dont les approximations successives biaisent suffisamment le résultat. Ainsi, il faut trouver un juste équilibre. C'est pourquoi ces 3% sont encore sujets de discussion même si ce choix nous semble raisonnable.

Néanmoins, cette plage a un souci. En effet, imaginons que la réponse a une valeur proche de zéro par exemple 0.02. Avec une telle valeur, la plage se résume donc à $[0.0198, 0.0202]$. Cette plage semble dès lors très sévère. Afin de pallier à ce problème, 1 centième de chaque côté est ajouté pour obtenir une plus grande plage. La plage est désormais définie:

$$[x - \frac{3 \cdot x}{100} - 0.01, x + \frac{3 \cdot x}{100} + 0.01] \tag{4.6}$$

Le décalage provoqué par ce terme 0.01 permet de créer une plage de minimum 0.02 autour de la réponse et cela peu importe sa valeur. Ce décalage de 0.01 nous semble raisonnable, car la plupart des exercices utilisent des composants dont les valeurs vont de l'unité à quelques dizaines au maximum. Pour des exercices utilisant principalement des valeurs bien plus grandes ou à l'inverse bien plus petites, la valeur de ce décalage peut être reconsidérée afin qu'il ne soit pas inutile dans le premier cas (aucun impact significatif sur la plage) ou inversement trop d'influence dans le second cas. Nous considérons que c'est raisonnable de fixer la valeur de ce biais à 1% de l'ordre de grandeur des valeurs en présence dans le circuit (par exemple dans notre situation 0.01, l'ordre de grandeur étant l'unité). Comme plage finale, on a donc désormais :

$$[x - \frac{3 \cdot x}{100} - 0.01 \cdot y, x + \frac{3 \cdot x}{100} + 0.01 \cdot y] \tag{4.7}$$

Où y est l'ordre de grandeur.

4.3 Conclusion

La simulation des exercices a été détaillée dans cette section. Les ressources utilisées par l'interface (frontend) sont converties en ressources pour la simulation (backend). Une fois ces ressources recupérées, NGSpice s'occupe de la simulation des circuits. Finalement, la correction et la structure des exercices sont explicitées.

Chapitre 5

Génération aléatoire

La génération de circuits aléatoires est une partie importante du mémoire d'un point de vue pédagogique. L'étudiant pourra s'entraîner sur des circuits électroniques à chaque fois différents, contrairement à aujourd'hui où le livre propose un nombre limité d'exercices. Il est aussi important de générer des circuits de difficultés variables pour s'adapter aux besoins de l'étudiant. Il s'entraînera d'abord sur des circuits simples et passera à des exercices plus complexes lorsqu'il se sentira à l'aise avec les exercices plus faciles. Pour gérer le niveau de difficulté du circuit, le professeur et les assistants pourront modifier le nombre de cycles du circuit et le type d'exercice généré :

- **Exercices directs** : l'étudiant doit calculer une tension ou un courant à partir du circuit.
- **Exercices indirects** : l'étudiant doit retrouver la valeur d'une impédance à l'aide d'une mesure de tension ou de courant du circuit.

En sachant que le nombre d'équations requis pour résoudre un circuit électronique augmente avec le nombre de cycles et que les exercices directs sont plus intuitifs que les indirects, on peut en déduire que les exercices simples auront peu de cycles et seront directs alors que les plus complexes auront un nombre de cycles plus élevé et seront indirects.

La génération aléatoire permet également aux professeurs et aux assistants de générer automatiquement un exercice pour une interrogation ou un examen. En effet, notre générateur simule l'exercice pour être sûr que la solution est unique et existe. Ceci évitera les surprises où plusieurs solutions sont en fait possibles dues à une indétermination.

5.1 Interface

Pour générer un circuit aléatoire, il faut spécifier les options de génération *Option random generation* entourées en rouge dans la figure 5.1 (dans l'onglet *Subproblems* de l'edit task). Comme expliqué auparavant, le nombre de cycles et le type d'exercice généré (direct ou indirect) peuvent être choisis. Une fois ces options choisies, il suffit de cliquer sur le bouton *Generate circuit* pour générer un nouveau circuit modifiable. Le nouveau circuit sera alors proposé aux étudiants après avoir sauvegardé les modifications.

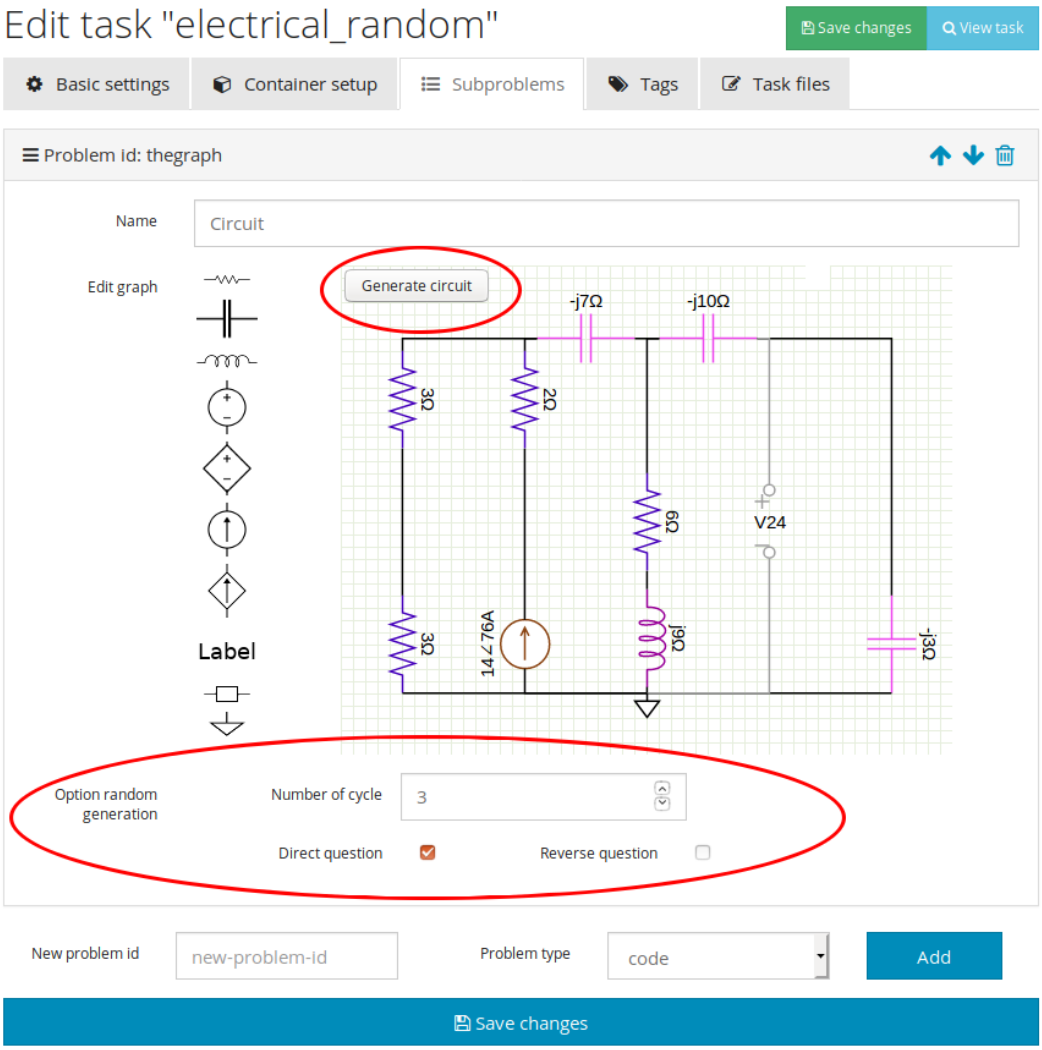


Figure 5.1: Choix des paramètres et génération du circuit

On peut aussi choisir de générer un nouvel exercice aléatoire à chaque fois que

l'étudiant relancera l'exercice INGInious. Pour ce faire, il faut spécifier un nombre positif *Random inputs* et cocher la case *Regenerate input random* dans l'onglet *Basic settings* comme montré dans la figure 5.2. Si ces paramètres sont donnés, le circuit de l'éditeur de l'onglet *Subproblems* n'est plus pris en compte et sera remplacé par un exercice aléatoire. Si *Regenerate input random* n'est pas coché, mais qu'un nombre est donné pour *Random inputs*, un circuit aléatoire sera généré, mais restera identique à chaque lancement d'INGInious.

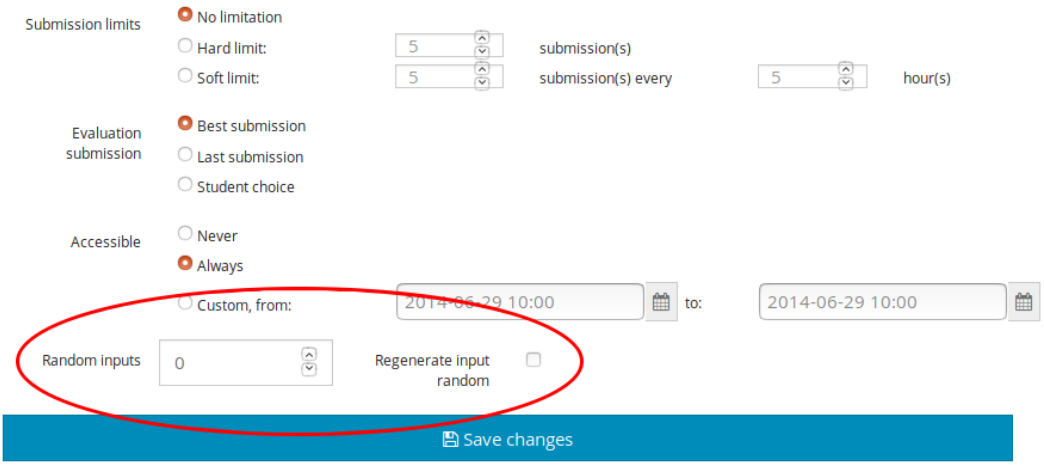
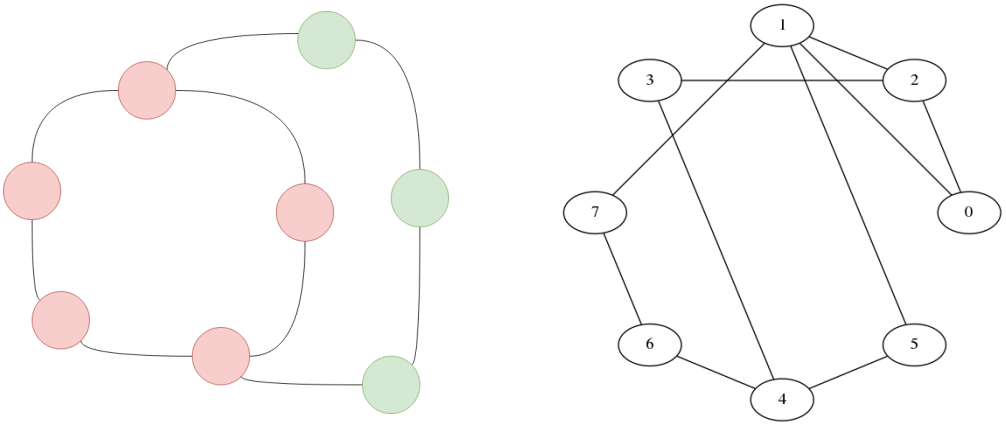


Figure 5.2: Regénérer un circuit à chaque lancement d'INGInious

5.2 Algorithme

5.2.1 Graph

Nous avons d'abord essayé d'implémenter la génération aléatoire avec l'aide d'une librairie de graphe. La librairie permet d'ajouter des noeuds et des arêtes et permet aussi de visualiser le graphe. Pour respecter le nombre de cycles donnés, un premier cycle est créé et de nouveaux cycles sont ajoutés au fur et à mesure. Cette façon de faire permet aussi d'avoir un graphe planaire. Un exemple est donné à la figure 5.3a où le cycle initial est en rouge et le cycle ajouté est en vert. Une fois le graphe formé, le résultat ressemble par exemple à la figure 5.3b.



(a) Ajout de cycle au graphe

(b) Exemple de graphe

Figure 5.3

Malheureusement, il ne suffit pas de remplacer les arêtes par des composants électroniques et les noeuds du graphe par les noeuds du circuit, car il faut aussi réorganiser le graphe pour avoir un circuit facilement lisible. Cette tâche s'avère relativement complexe, les circuits peuvent facilement être représentés sous forme de graphe, mais dessiner un circuit lisible à partir d'un graphe est difficile.

Pour pouvoir dessiner le graphe sous une disposition orthogonale, l'idée la plus simple est d'utiliser une grille qui représente les possibles positions des composants et des fils. Les cycles élémentaires sont dessinés un à un jusqu'à dessiner la totalité du graphe. Malheureusement, la grille est limitée dans l'espace et la disposition de ces cycles pour ne pas "dépasser" la grille devient rapidement un casse-tête. En effet, seulement un cycle peut être dessiné selon une multitude de façon comme montré à la figure 5.4. De plus, le choix de disposition d'un cycle influence la disposition des autres cycles qui pourraient ne plus avoir de place.

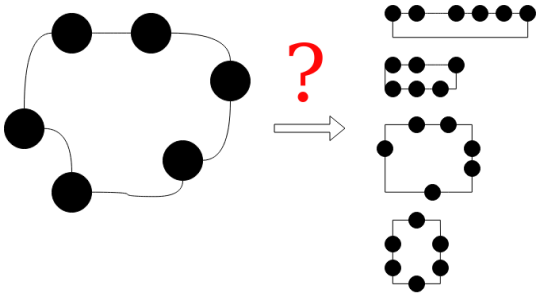


Figure 5.4: Dessiner un graphe orthogonal

Aussi, la structure de graphe est pratique pour parcourir les noeuds, trouver un chemin minimum, trouver un flot maximum ou minimum, faire une recherche,

etc. Ici, nous voulons simplement dessiner un circuit électronique, la structure de graphe n'est pas très utile.

C'est pourquoi nous avons plutôt opté pour une solution heuristique qui attribue directement les emplacements à utiliser des composants et des fils. Ce qui évite de devoir transformer ultérieurement un graphe en un circuit électronique.

5.2.2 Heuristique

La stratégie utilisée s'inspire fortement de la création du graphe précédemment expliquée : un premier cycle est créé comme avant pour ensuite y ajouter d'autres cycles. Mais contrairement au graphe, les positions prédéfinies sont utilisées en même temps que la création aléatoire. Ces positions et les fils associés sont montrés dans la figure 5.5.

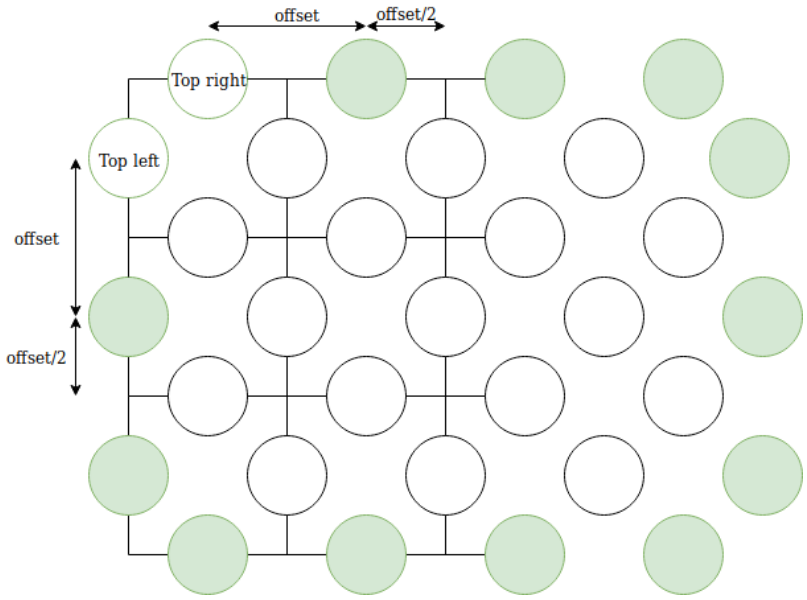


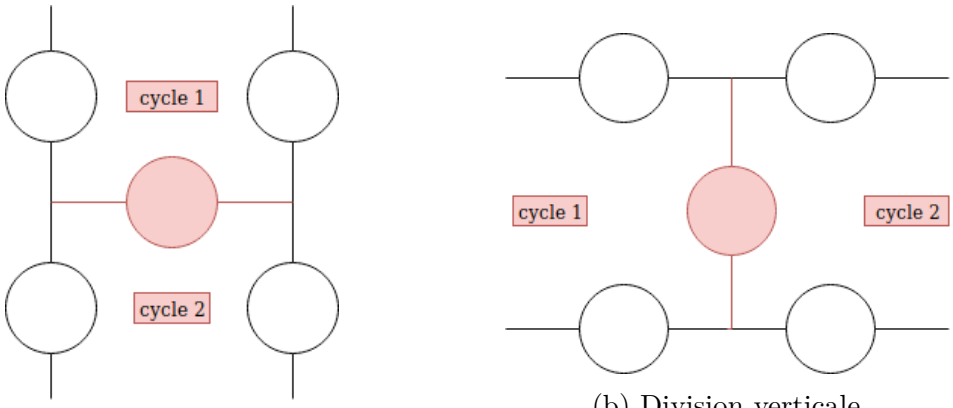
Figure 5.5: Tous les emplacements possibles des composants

Au total, 31 positions sont possibles, soit 44 cycles internes différents possibles, ce qui est largement suffisant pour faire des circuits simples ou complexes et avoir une grande diversité de circuits. Toutes ces positions sont définies grâce aux coordonnées *top right*, *top left* et grâce à *offset*.

Le premier cycle est formé des positions externes montrées en vert sur la figure 5.5. Chaque cycle est représenté par quatre listes de composants qui correspondent aux composants du dessus, de droite, du dessous et de gauche du cycle. Ces listes sont ordonnées pour placer les fils plus facilement. Pour le premier cycle, les

emplacements du dessous sont réservés pour placer la masse et les autres côtés doivent contenir au moins un composant pour faciliter l'ajout de cycle par la suite.

Une fois le premier cycle formé, celui-ci va être divisé horizontalement ou verticalement pour former deux nouveaux cycles à la place de l'ancien cycle et ainsi de suite jusqu'à avoir le nombre de cycles voulu. Chaque division va donc ajouter un nouveau cycle interne comme montré dans la figure 5.6.



(a) Division horizontale

(b) Division verticale

Figure 5.6: Divisions possibles

Pour trouver une nouvelle division, il faut d'abord choisir un cycle au hasard dans le circuit. Ensuite, pour une division horizontale, on choisit aléatoirement un composant appartenant à la liste des composants de gauche du cycle pour l'attacher (en ajoutant de nouveaux composants) à un composant à droite du cycle. Le second composant est choisi de manière à ce que la division ne "passe" pas sur d'autres composants (voir figure 5.7). De même, pour une division verticale, on attache un composant du dessus avec un composant du dessous. Bien sûr, il faut vérifier au préalable que la division est possible pour ce cycle: des composants doivent être présents de chaque côté et il doit y avoir au moins un nouvel emplacement disponible pour insérer la division.

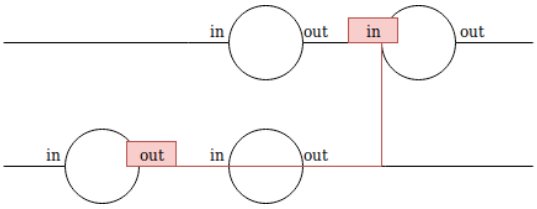
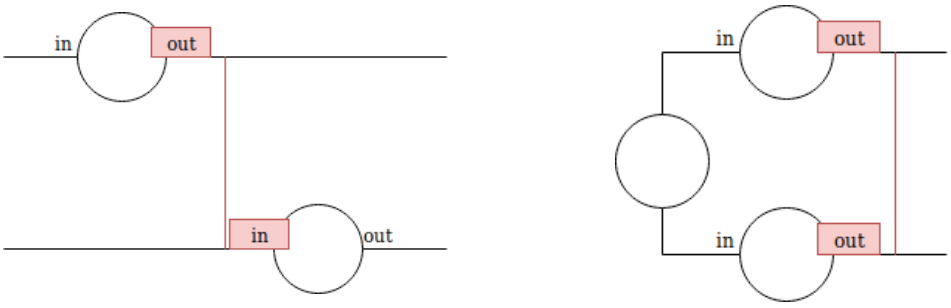


Figure 5.7: Choix de composant non possible

Pour insérer la nouvelle division, les ports doivent être choisis prudemment.

Trois cas sont à considérer :

- Les deux composants à connecter sont à la limite du cycle comme montré sur la figure 5.8b. Les ports doivent se trouver de l'autre côté de la limite.
- Un composant est décalé par rapport à l'autre comme montré sur la figure 5.8a. Les ports doivent se trouver à l'intérieur des deux composants.
- Les deux composants sont alignés, les ports peuvent être choisis librement tant qu'ils sont du même côté.



(a) (b)

Figure 5.8: Choix des ports

Pour l'instant, seules des impédances (résistances, capacités et inductances) ont été ajoutées au circuit. Des sources de tensions ou de courant sont maintenant ajoutées en remplaçant des impédances tout en faisant attention de remplacer aussi les ports.

Il n'y a plus qu'à ajouter la question, selon le type d'exercice choisi:

- Pour une question directe sur une tension, on ajoute une nouvelle division qui contient uniquement un label de tension. La division est reliée à la masse pour respecter le format d'exercice du livre.
- Pour une question directe sur un courant, une impédance est remplacée (comme pour ajouter les sources) par un label de courant en faisant bien attention de ne pas court-circuiter le circuit. Un court-circuit est la mise en connexion de deux points du circuit électrique par un conducteur de faible résistance, par exemple un fil sans composant, qui pourrait rendre l'exercice non solvable. Il suffit de vérifier qu'il y a au moins deux composants sur le fil pour pouvoir remplacer l'un des deux sans créer un court-circuit.

- Pour une question indirecte, un label de tension ou de courant est d'abord ajouté. Le circuit est ensuite simulé pour déterminer la valeur du label. La valeur du label est affichée sur le circuit et une des impédances est cachée.

Pour que l'exercice soit pédagogique et faisable, certaines valeurs de label doivent être prohibées. Comme montré dans la figure 5.9, des labels placés aléatoirement peuvent produire des exercices dont la réponse est évidente ou alors pire, rendre l'exercice insolvable pour les questions indirectes (aucune information supplémentaire pour résoudre les équations). De plus, certains circuits ne sont pas compatibles avec la résolution mathématique de NGSpice. Une inductance seule placée en série avec une source de tension empêche la convergence des valeurs de NGSpice. La simulation permet donc aussi de vérifier le circuit.

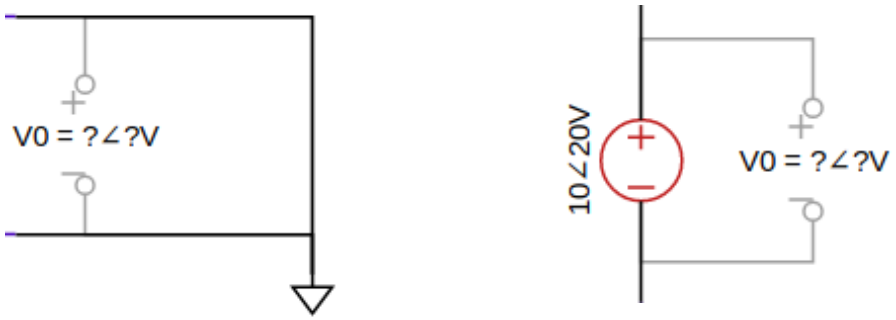


Figure 5.9: Mauvais labels

Pour finir, le moteur de template Jinja a été utilisé pour passer de la logique Python (dictionnaire des composants et des fils) à la structure **XML** du circuit électronique. Un pseudocode est donné ci-dessous pour résumer le déroulement de la génération.

Algorithm 1 Génération d'un circuit aléatoire

```
cycles ← ∅ //ensemble des cycles du circuit
i ← 0
while i ≠ nbrCycle do
  if i == 0 then
    cycleExtérieur ← nouveau cycle sur le bord de la grille
    cycles ← cycles ∪ cycleExterieur
  else
    cycle ← tirage aléatoire(cycles)
    cycle1, cycle2 ← division(cycle)
    cycles ← cycles ∪ cycle1 ∪ cycle2
  end if
  i ← i + 1
end while
Ajout d'une question
Ajout d'une source
return XML avec Jinja
```

5.3 Faisabilité des exercices inverses

Cacher un composant pour créer un exercice inverse peut produire un exercice infaisable. Nous allons montrer dans quels cas un exercice est infaisable ou ne l'est pas, à l'aide d'un théorème et d'une hypothèse.

Théorème 1 Un système d'équations linéaires indépendantes à n inconnues possède une unique solution lorsque le nombre d'équations est égal à n [13].

Hypothèse 1 Tous les courants et tensions des circuits électroniques concernés peuvent être déterminés grâce à une loi des mailles/noeuds appliquée sur chaque maille/noeud du circuit.

D'après le théorème 1 et l'hypothèse 1, on peut trouver une solution unique de tension ou de courant grâce à un système d'équations linéaires à n équations où n est le nombre d'inconnues (tension ou courant). Ce système d'équations découle de la loi des mailles/noeuds.

Si on cache un composant, on ajoute une inconnue au système d'équations. En effet, chaque terme d'une des lois de Kirchhoff correspond à la valeur de tension ou de courant d'un composant. Pour que ce système possède toujours une unique solution (autant d'inconnues que d'équations), il faut ajouter une nouvelle équation indépendante des équations déjà présentes. Pour ce faire, on ajoute une mesure de tension ou de courant sur le circuit. La mesure doit être évidemment utile, si une

mesure correspond à la valeur d'une source, l'information n'ajoutera pas d'équation indépendante.

Si on cache plusieurs composants, on doit ajouter autant de nouvelles équations que de composants cachés (toujours pour avoir le même nombre d'équations que d'inconnues). Cependant, les mesures ajoutées au circuit doivent aussi être indépendantes entre elles. C'est-à-dire qu'une mesure ne doit pas être déductible d'une autre mesure sinon les équations ajoutées ne seront pas indépendantes.

Dans notre cas, seulement un composant peut être caché pour simplifier la recherche d'exercices faisables. Il suffit alors d'ajouter une mesure utile pour que l'exercice reste possible.

5.4 Conclusion

La génération aléatoire permet de produire une infinité de circuits électriques. Les circuits générés sont faisables et peuvent avoir des niveaux de difficulté variables selon les paramètres choisis.

Chapitre 6

Retour

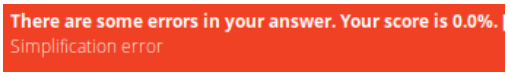
L'application doit être capable de vérifier l'exactitude d'une réponse donnée à un exercice mais doit aussi être capable de donner des retours (feedback) pertinents. En effet, si un étudiant se trompe dans un exercice, un simple retour négatif ne lui procurera aucune piste pour se corriger et s'améliorer. L'application n'aurait pas beaucoup d'intérêt pédagogique (un recueil d'exercices papier aurait tout simplement la même efficacité) sans retour constructif à part créer ou générer des circuits électroniques. C'est pourquoi un environnement qui tente d'être plus pédagogique a été créé.

6.1 Fonctionnement

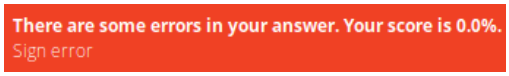
Pour commencer, l'exactitude de la solution finale est vérifiée. Si elle est correcte, l'étudiant a réussi l'exercice et aucun retour n'est à fournir. Si elle est incorrecte, deux erreurs courantes sont vérifiées :

- Si un changement de signe que ce soit pour la magnitude ou pour la phase conduit à une bonne solution, l'étudiant a commis une erreur de signe.
- Si la solution se trouve dans une plage à plus de 3% de la solution, mais à moins de 10% de la solution, l'étudiant a probablement commis une erreur de simplification. Cependant, des calculs faux pourraient par chance être proches de la bonne solution. Des améliorations possibles seraient alors d'obliger l'étudiant à écrire un minimum d'équations correctes avant de vérifier une erreur de simplification.

Ces deux erreurs sont ajoutées au retour comme on peut le voir dans la figure 6.1. Mais si l'étudiant n'a commis aucune de ces deux erreurs facilement détectables, un meilleur moyen de retour doit être mis en place.



(a)



(b)

Figure 6.1: Erreur dans la solution finale

Pour trouver un bon moyen d'indiquer les erreurs à l'étudiant, il faut d'abord comprendre comment les exercices sont résolus. Les solutions des exercices traités par l'application peuvent être trouvées grâce à un système d'équations. Un moyen de donner un bon retour aux étudiants est donc de leur permettre d'inscrire leurs équations et de les vérifier une par une pour ensuite donner un retour pour chacune d'entre elles. Ce système d'équations peut contenir des équations résultantes de la loi des mailles ou de la loi des noeuds[14].

La loi des noeuds stipule que *la somme algébrique des intensités des courants qui entrent par un noeud est égale à la somme algébrique des intensités des courants qui en sortent.* Par exemple, une loi des noeuds appliquée au schéma 6.2 donne : $i_2 + i_3 = i_1 + i_4$.

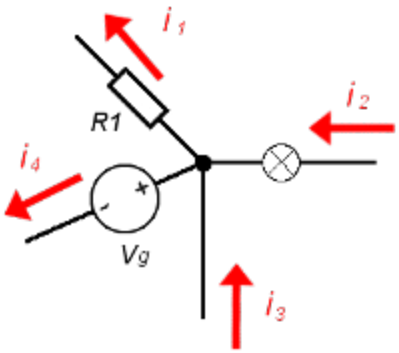


Figure 6.2: Illustration de la loi des noeuds

L'étudiant peut ajouter un nouveau nom de courant pour chaque composant ce qui lui permet d'utiliser avec beaucoup de liberté cette loi. Il peut aussi ajouter un nom de tension n'importe où sur les fils pour créer par exemple l'équation : $i_2 + i_3 = V_1/R1 + i_4$.

La loi des mailles est la deuxième loi de Kirchhoff. Cette loi stipule que *dans une maille d'un réseau électrique, la somme des tensions le long de cette maille est toujours nulle.* De la même manière, l'étudiant peut ajouter des nouveaux noms de courant ou de tension pour utiliser cette loi avec beaucoup de liberté. Par exemple, une loi des mailles appliquée au schéma 6.3 donne : $V_1 - V_2 - V_3 - V_4 = 0$ ou aussi $V_1 - i_2 R_1 - V_3 - V_4 = 0$.

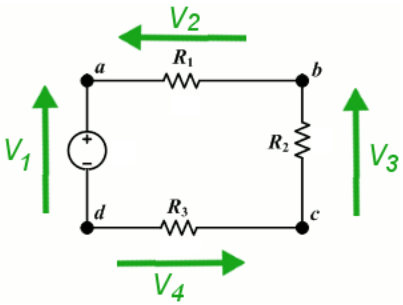


Figure 6.3: Illustration de la loi des mailles

Avec la possibilité d'utiliser ces deux lois et d'écrire des nombres imaginaires, l'étudiant peut écrire son système d'équations où chaque équation sera vérifiée (voir figure 6.4) si sa solution finale n'est pas correcte. L'étudiant pourra donc trouver dans quelle équation son erreur se trouve et la corriger. Pour corriger son erreur, il pourra aussi subdiviser l'équation incorrecte en plusieurs nouvelles équations pour trouver encore plus facilement son erreur. En effet, la forme des équations est très permissive et beaucoup d'équations peuvent être ajoutées ce qui laisse une grande liberté à l'étudiant pour bien comprendre le circuit.

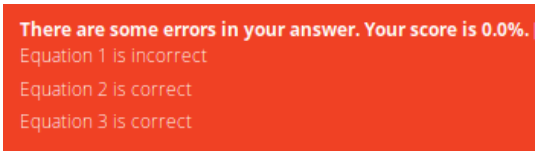


Figure 6.4: Vérification de chaque équation

En plus de vérifier l'exactitude des différentes équations soumises par l'étudiant, l'application essaye de détecter une erreur courante, l'erreur de signe. Comme vous avez pu le voir précédemment, les équations définies par les lois de Kirchhoff produisent plusieurs termes. Pour chacun de ces termes, une erreur de signe est envisageable. Lorsqu'une équation est déterminée incorrecte, l'application va vérifier si un ou plusieurs changements de signe des termes peuvent aboutir à une équation correcte. Si c'est le cas, l'étudiant a commis une erreur de signe et recevra un retour comme à la figure 6.5.

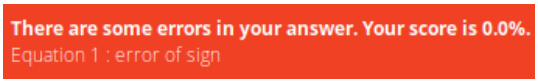


Figure 6.5: Vérification du signe des termes

6.2 Implémentation

Le code des retours se trouve dans plusieurs fichiers. La réponse finale de l'étudiant est comparée à la valeur de la simulation Spice dans le fichier *parser_ngspice.py*. Le parser retourne *true* si la réponse est correcte et *false*, *simplification* ou *signe* en fonction de l'erreur de l'étudiant. Pour une impédance, la réponse de l'étudiant est simplement comparée à la valeur cachée du composant.

La valeur de retour est passée au fichier *run* de l'exercice qui lance *parser_equation.py* si l'étudiant n'a pas trouvé la bonne solution. Ce second parser s'occupe des équations de l'étudiant. Pour ce faire, les équations ainsi que leurs ids sont récupérées individuellement grâce aux tags *NextEquation* et *ID*. Les valeurs des symboles par exemple V_1 sont récupérées dans un dictionnaire pour les remplacer dans les équations par leurs valeurs numériques respectives. Les symboles imaginaires i ou j sont remplacés par I, le symbole utilisé par la librairie Sympy de python. En effet, Sympy est utilisé pour nous aider dans l'évaluation des équations. Nous utilisions au début simplement la fonction *eval* de python pour évaluer les équations, mais cette façon de faire peut poser des problèmes de sécurité puisque si l'étudiant écrit du code python, celui-ci sera exécuté. Les nombres du type magnitude$\angle$phase sont remplacés par un nombre imaginaire de type $a + bi$ car la simulation Spice retourne des valeurs de ce type. Finalement, le membre de gauche est soustrait au membre de droite. Si le résultat est proche de zéro, l'équation est correcte. On pourrait simplement vérifier l'égalité du membre de gauche au membre de droite, mais cette technique ne permet pas de laisser une tolérance aux erreurs d'arrondis.

Si l'équation n'est pas correcte, les signes des termes sont modifiés pour détecter une erreur de signe de la part de l'étudiant. Pour avoir toutes les possibilités, une méthode récursive *sign_checker* parcourt un arbre (dfs) comme montré à la figure 6.6.

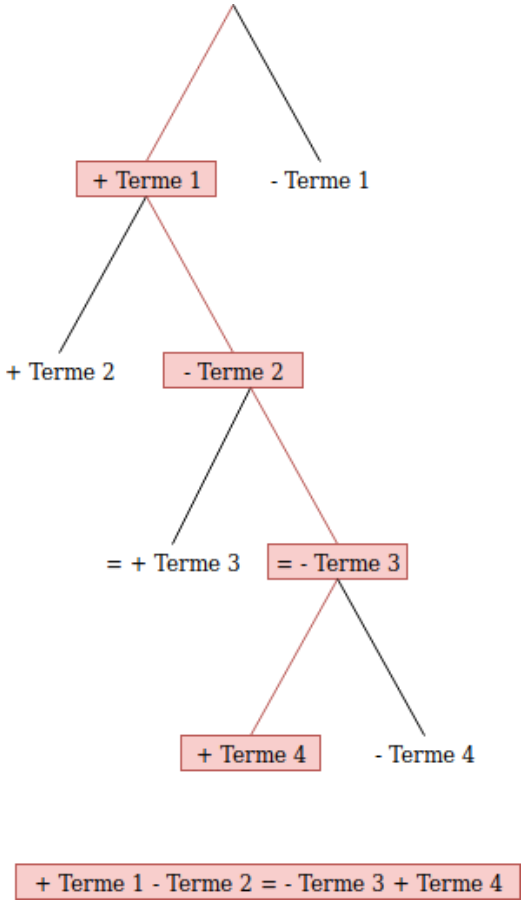


Figure 6.6: Parcours des choix de signes

Sur cette figure, un exemple de chemin dans l'arbre est représenté en rouge. L'équation est divisée pour avoir chaque terme individuellement. Le chemin attribue à chacun de ces termes un signe pour aboutir à une équation qui pourrait être correcte si l'étudiant s'est trompé dans l'un de ses signes.

6.3 Conclusion

Dans le cadre des exercices implémentés, donner un retour pertinent n'est pas une chose facile. En effet, le retour peut très facilement être inutile soit parce qu'il n'aide aucunement l'étudiant soit parce que l'aide résout l'exercice à sa place. De plus, les exercices impliquent fréquemment une résolution en plusieurs étapes. C'est pourquoi il est presque impossible, à partir de la réponse seule, de trouver l'étape qui a causé l'erreur. Les outils permettant à l'étudiant d'exposer sa résolution réussissent à contourner ces problèmes. En effet, en analysant sa

résolution, INGInious arrive à pointer précisément là où l'erreur à été commise.

Chapitre 7

Evaluation des étudiants

7.1 Démarche

Trouver des lacunes, des failles dans la plateforme ou des pistes d'améliorations n'est pas chose aisée. D'un point de vue du concepteur, toutes les fonctionnalités semblent évidentes à utiliser, en effet les ayant implémentées, il sait comment elles fonctionnent. De plus, les multiples failles et divers bugs qui résident encore dans l'application ne lui sautent pas aux yeux. En effet, si c'était le cas, elles n'existeraient plus. C'est pourquoi un avis extérieur est essentiel afin de corriger les divers soucis et de proposer des améliorations opportunes (tout comme des tests unitaires).

Qui mieux pour juger le plugin circuit électrique pour INGInious que ceux qui seront possiblement amenés à l'utiliser? C'est ainsi que l'outil a été installé sur un serveur pour qu'il soit disponible à tous et notamment aux étudiants. Le site[1] contient déjà une série de questions issues du livre *Engineering Circuit Analysis* [15] et d'anciens examens des cours **LFSAB1201 - Physique 1** et **LELEC1370 - Circuits et Mesures**. Des exercices générés aléatoirement avec des difficultés croissantes ont également été ajoutés :

- **Facile** : exercices directs avec peu de cycles
- **Intermédiaire** : exercices directs avec plus de cycles
- **Difficile** : exercices indirects

Notre promoteur Claude Oestges, qui est également le professeur du cours de Circuits et mesures, nous a permis de tester notre outil dans un premier temps

[1] `https://tfe-arthurs.info.ucl.ac.be/courselist`

avec les assistants et dans un second temps avec les étudiants du deuxième bloc annuel suivant ce cours.

Les assistants nous ont donné un premier retour sur la qualité de l'application. Quelques modifications visuelles étaient à améliorer comme l'ajout d'une unité manquante ou encore quelques coquilles concernant certains termes utilisés (par exemple l'utilisation du mot "value" à la place de magnitude, etc). Mais surtout, la prise en main de l'application n'était pas assez intuitive pour un nouvel utilisateur. Dans la perspective de combler ce défaut, une vidéo explicative a été enregistrée et mise en ligne. Cette vidéo tutorielle détaille l'utilisation des différentes fonctionnalités. La plus évidente, bien entendu, comment encoder et soumettre sa réponse. Mais également, comment annoter le circuit via des labels de noeud et/ou de courant et utiliser ces annotations afin de créer des équations. Cette vidéo a été partagée via Youtube[1] .

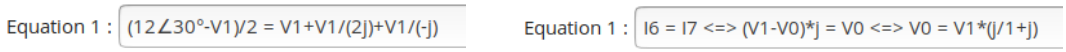
Pour avoir un retour des principaux concernés, les étudiants, un formulaire[2] , a été partagé. Le formulaire contient une série de questions sur divers aspects de l'application. Ceci nous permet d'avoir une évaluation sur la qualité et l'utilité du travail. Malheureusement, peu d'étudiants prennent le temps de travailler sans crédits à la clé ou une quelconque contrepartie. Un système de tirage au sort a donc été mis en place parmi les participants qui ont réussi plusieurs exercices pour les motiver à tester, à s'entraîner, à résoudre et finalement à donner leur avis.

7.2 Analyse des anciennes soumissions

De plus, grâce aux anciennes soumissions du plugin sur INGInious, toutes les réponses des étudiants (une trentaine) peuvent être visualisées et examinées. Avec ces informations, les problèmes récurrents des utilisateurs peuvent être détectés. Certains étudiants ont la bonne intuition et arrivent à utiliser la totalité ou une partie des fonctionnalités (ajouter des équations, des noms de labels et encoder la solution). Mais d'autres étudiants ne comprennent pas complètement le fonctionnement et les limites de l'application. Par exemple, les équations requièrent un format adéquat. Ainsi si la plupart des symboles (comme les opérations élémentaires +, *, - , /) sont pris en charge, ce n'est pas le cas de tous. Ainsi, certaines équations utilisent des symboles particuliers comme à la figure 7.1 ce qui empêche Sympy de l'interpréter.

[1] `https://www.youtube.com/watch?v=HoJV1DdnwfQ&t=43s`

[2] `https://docs.google.com/forms/d/e/1FAIpQLSelFRO4cuVe3SWQEq3URzViyR9JuKt_qhwIK2Vi-AutrkFlbg/viewform`

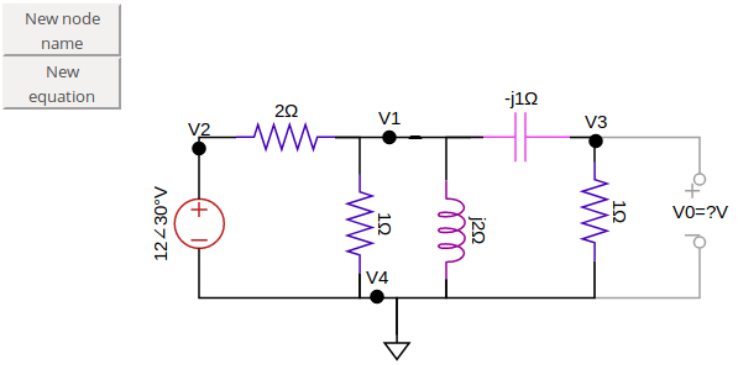


(a) (b)

Figure 7.1: Mauvais encodage d'équation

Pour ces deux équations, les étudiants utilisent des symboles non pris en charge par notre application, l'équation 7.1a utilise le symbole ° pour signifier l'unité des degrés et l'équation 7.1b utilise le symbole <=> pour signifier une équivalence. L'encodage des équations est montré dans la vidéo. Néanmoins, la présence de ces erreurs est une preuve que ce n'est pas suffisant. Ces soucis d'encodage n'entraînent pas une mauvaise correction de l'exercice, mais la correction de l'équation (et de celles qui suivent) est dès lors perturbée et donc le feedback également. Une solution complémentaire serait d'interdire tous les symboles non pris en charge par l'application dans les équations et de prévenir les caractères qui posent soucis. Les nombres, les opérateurs mathématiques (+, -, *, /), le signe phaseur et un signe égal seraient alors les seuls symboles disponibles.

Un autre exemple est donné à la figure 7.2.



Equation 1 : (V2-V1)/2= (V1-V4)/1 + (V1-V4)/(2*j) + (V1-V3)/(-j)

Equation 2 : (V1-V3)/(-j) = V3

Equation 3 : V2-V4 = 12∠30

Equation 4 : V0 = V3-V4

Equation 5 : (V1-V3)/(-j) = (V3-V4)/1

Figure 7.2: Pas de solution

L'étudiant a bien ajouté des noms de labels et les utilise correctement dans ses équations qui sont toutes correctes. Mais il n'encode pas la solution finale. De même, ceci est expliqué dans la vidéo explicative. Une autre solution serait d'indiquer à l'étudiant qu'il n'a pas répondu à la question au moment où il souhaite soumettre sa réponse. Cette fonctionnalité est intégrée à INGInious et fonctionne si l'étudiant n'a pas écrit d'équation dans un des champs prévus à cet effet, mais ne fonctionne pas pour la solution finale. Si l'étudiant soumet sans répondre, la correction évalue l'exercice à 0% et le feedback le prévient de son oubli.

7.3 Résultats du formulaire

Avec les différentes réponses du formulaire (une dizaine d'étudiants), des statistiques ont pu en être extraites. Le formulaire contient neuf questions que voici :

Question 1 : L'interface est-elle lisible?

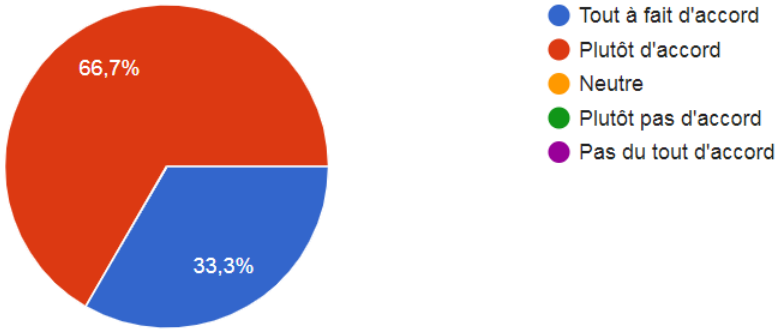


Figure 7.3: Résultats de la Question 1

Il n'y a aucun retour négatif ou mitigé concernant la lisibilité de l'interface. Il semble ne pas avoir de gros soucis de lisibilité à régler.

Question 2 : L'outil est facile d'utilisation?

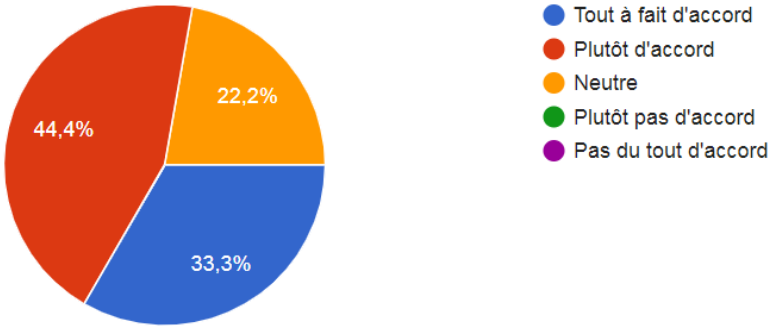


Figure 7.4: Résultats de la Question 2

Globalement, l'outil semble agréable à prendre en main. Néanmoins, d'après les anciennes soumissions, un nombre important d'étudiants n'utilisent pas toutes les fonctionnalités (annotations, équations) et se limitent à répondre directement à la question ce qui est bien évidemment plus facile que l'ensemble des fonctionnalités.

Question 3 : Le feedback généré par INGInious permet-il de trouver son erreur?

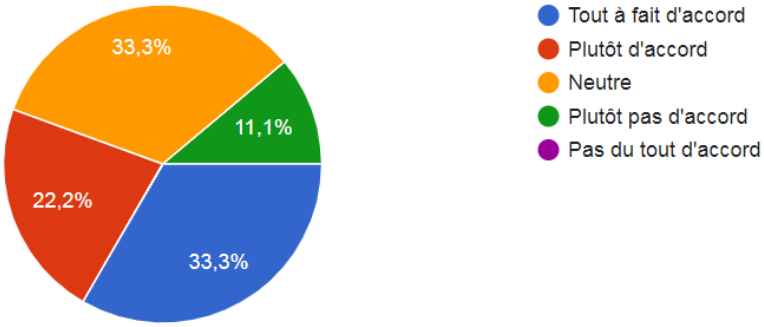


Figure 7.5: Résultats de la Question 3

Les résultats concernant la question 3 sont plus mitigés. Une légère majorité considère le feedback comme utile néanmoins une partie non négligeable des avis sont plus partagés. De nouveau, il faut également mettre en perspective que certains étudiants n'ont que très peu utilisés le système d'annotations et d'équations de l'outil alors que le système de feedback se base notamment sur ce système. Améliorer le feedback et mettre en avant le système d'annotations du circuit sont, suivant ces résultats, des pistes d'améliorations.

Question 4 : L'application peut-elle apporter une aide à la compréhension des exercices?

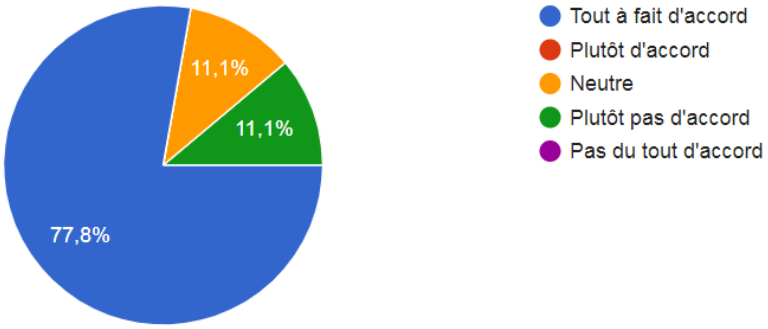


Figure 7.6: Résultats de la Question 4

La majorité des étudiants pense que la plateforme peut aider à la compréhension des exercices et ainsi être un support supplémentaire pour réussir les cours d'électricité.

Question 5 : L'application est-elle agréable à utiliser?

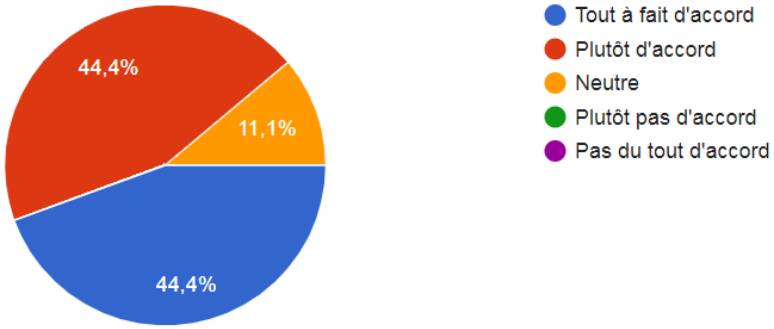


Figure 7.7: Résultats de la Question 5

Aucun problème par rapport à ce sujet. Les retours sont positifs et l'application semble agréable d'emploi.

Question 6 : Les exercices aléatoires permettent-ils de s'entraîner aux examens?

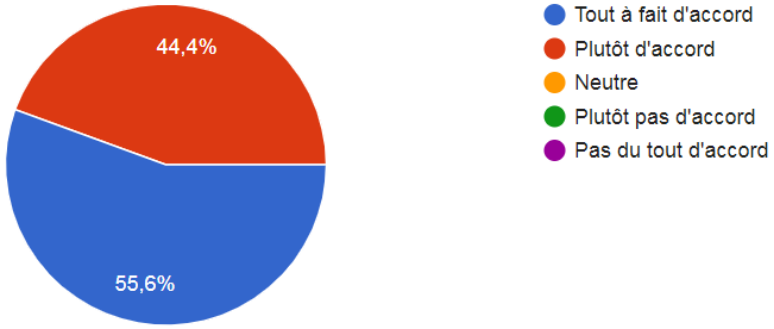


Figure 7.8: Résultats de la Question 6

Comme à la question 4, l'application paraît convaincre les étudiants de son utilité dans la réussite des cours.

Question 7 : Les exercices aléatoires donnent-ils des questions triviales (par exemple le courant à la sortie d'une source de courant)?

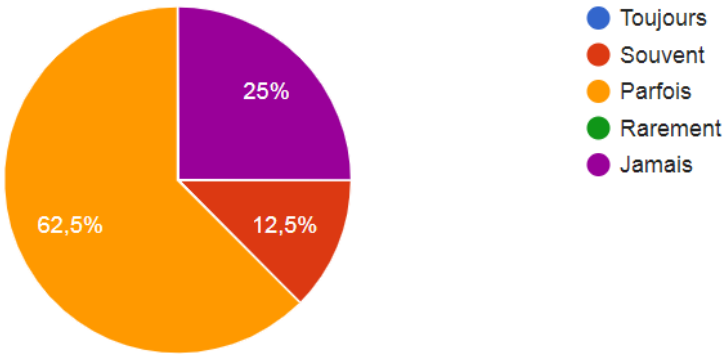


Figure 7.9: Résultats de la Question 7

Question 8 : Une interrogation sur l'application ne me dérangerait pas.

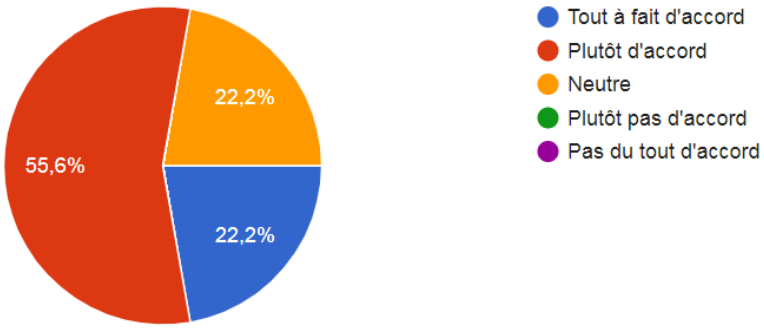


Figure 7.10: Résultats de la Question 8

L'application comme support d'évaluation est également jugée positivement par les étudiants. Cela signifie qu'ils font confiance au plugin pour les évaluer.

Question 9 : La vidéo explicative est-elle suffisante pour prendre en main l'application?

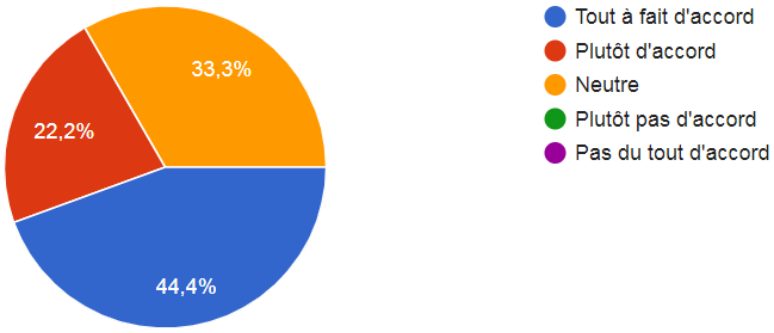


Figure 7.11: Résultats de la Question 9

La vidéo de démonstration de l'application semble suffisante pour utiliser l'application et manipuler les différents outils de celle-ci. Il n'y a apparemment aucun souci de prise en main pour un nouvel utilisateur.

7.4 Conclusion

Cette section nous a permis de nous rendre compte du ressenti général des utilisateurs. Globalement, les avis sont très positifs ce qui confirme l'utilité de l'extension pour les étudiants. Ces avis ont aussi mis en évidence que le feedback n'aide pas toujours efficacement les étudiants à trouver leurs erreurs. Cela provient probablement du fait que l'outil d'annotations et d'inscription d'équations n'est pas assez mis en avant et donc pas assez utilisé. De plus, l'analyse des anciennes soumissions et des différentes interactions des utilisateurs avec l'application ont permis la découverte de failles et de dysfonctionnements qui ont été résolus depuis lors. Les anciennes soumissions montrent aussi que les feedbacks permettent à l'étudiant de trouver son erreur lorsque l'outil est bien utilisé.

Chapitre 8

Améliorations et extensions

8.1 Améliorations des outils implémentés

Le système de feedback peut encore être amélioré afin de cibler de manière encore plus précise les possibles erreurs. Par exemple, en identifiant deux équations équivalentes ou alors en identifiant certaines équations qui n'ont pas réellement de sens par rapport aux lois de Kirchhoff.

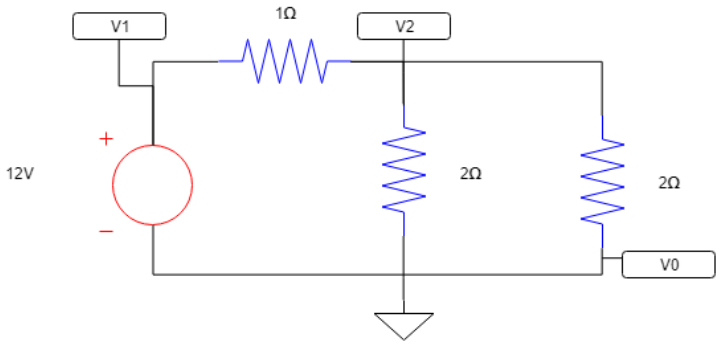


Figure 8.1: Circuit

Par exemple, pour le type de circuit représenté à la figure 8.1, l'équation $12 = (V1 - V2) + (V2 - V0)$ est vérifiée. Néanmoins, l'équation $12 = (V1 - V2) + (V2 - V0) + 134 \cdot V0$ l'est aussi. Le terme $134 \cdot V0$ n'a bien évidemment pas de sens ici. Cela vient du fait qu'actuellement, le correcteur évalue les équations numériquement et $V0$ étant nul, l'équation demeure correcte. Néanmoins, correct ou non, ce terme superflu montre probablement une incompréhension des lois qui régissent les circuits. Rajouter une évaluation symbolique au correcteur permettrait de mettre en évidence ce type d'erreur.

Un interpréteur du format LaTeXpeut également être ajouter dans le champ d'inscription des équations afin d'avoir une mise en page plus propre.

8.2 Extensions et améliorations dans le domaine de l'électricité

Notre implémentation permet de simuler un groupe d'exercices précis: utiliser la loi de Kirchhoff pour trouver une mesure ou une valeur de composant, pour des circuits résistifs ou complexes. Cependant, une multitude d'autres exercices existe et pourrait être ajoutée:

- Circuits avec transistors
- Calcul de l'impédance d'entrée et de sortie d'un circuit
- Circuit triphasé (convertisseurs)
- Tracer un diagramme de bode
- Équivalent de Thévenin/Norton
- Modélisation d'un filtre
- Circuit magnétique (bobines magnétiquement couplées)
- Etc.

Avec de légères modifications (ajout de composants, commande de simulation modifiée) de l'extension, de nouveaux types d'exercices électriques pourraient être créés. De plus, de nouveaux types d'exercices demandant la réalisation ou le dimensionnement de circuits peuvent dans le futur être implémentés. Cette catégorie d'exercices pourra potentiellement servir de base pour ensuite utiliser d'autres logiciels de circuits schématiques comme LTspice.

8.3 Extensions vers d'autres domaines

L'utilisation de la librairie MxGraph permet d'étendre INGInious vers de nombreux autres domaines. Des exercices de connectivités entre routers peuvent par exemple être imaginés pour le cours LINGI1341-Réseaux informatiques [16].

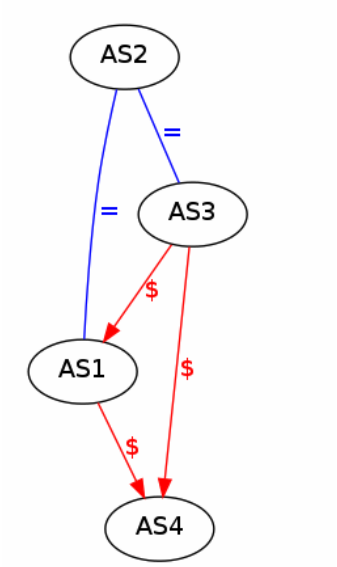


Figure 8.2: Schéma d'un exercice typique issu du cours Réseaux informatiques

La Figure 8.2 montre une topologie réseau d'un exercice où l'objectif est de déterminer les tables de routage des différents ASs. Le cours LINGI2146 Mobile and Embedded Computing[17] est également propice à ce type d'exercices de "routing".

Les cours d'algorithmes [18] peuvent également profiter de la visualisation, notamment dans l'apprentissage d'algorithme de recherche dans un graphe tel que le Breadth First Search ou le Depth First Search.

8.4 Conclusion

Cette extension est peut être le début d'un grand nombre de nouveaux exercices, car le potentiel de mxGraph dans INGInious est énorme et ouvre la voie à des applications diverses et variées.

Chapitre 9

Conclusion

INGInious est un fabuleux outil pédagogique qui à l'heure actuelle permet déjà de fournir une formation et une évaluation de compétences à des centaines d'étudiants de l'Université catholique de Louvain. Le travail accompli a pour objectif d'étendre le nombre de compétences et d'agrandir le nombre d'étudiants concernés par la plateforme.

Les aptitudes visées par le travail ont rapport à la physique des circuits électriques. En toute logique, les étudiants dont les options ont un rapport avec ces aptitudes sont le public cible même si nous espérons que n'importe qui puisse être intéressé par les outils développés.

En conclusion, notre mémoire a quatre objectifs clairs :

- Fournir une interface claire et lisible.
- Être capable de corriger des exercices sur les circuits électriques.
- Fournir des outils pour exposer sa résolution de l'exercice et ensuite fournir un feedback pertinent.
- Prodiguer un nombre important d'exercices.

Il est difficile de juger de la qualité de l'interface, ce point étant très subjectif et sujet à des appréciations diverses. C'est pourquoi obtenir le retour des utilisateurs est primordial sur ce point afin de valider s'ils sont positifs et de rectifier la mise en forme dans le cas contraire.

Les exercices implémentés se basent principalement sur la résolution de circuit à l'aide de phaseur. Dans ces circuits, la majorité des composants de base sont implémentés et utilisables. Les réponses de l'utilisateur sont vérifiées auprès des résultats de la simulation. Basé sur NGSpice, le plugin requiert certaines précautions afin de garantir son bon fonctionnement. Aucune topologie connue ne donne de

résultats incorrects à l'exception bien évidemment des topologies qui n'ont pas de logique physique.

Différents outils sont également disponibles pour exposer sa résolution. Un système complet d'annotations de circuit afin d'indiquer courant et tension est disponible. Il y a également la possibilité d'évaluer et de vérifier différentes équations afin de progresser par étapes et de plus facilement déterminer le moment où une erreur a été commise.

Finalement concernant la quantité d'exercices fournis, une section d'édition est dédiée à la création de circuits. Grâce à cet outil d'édition, un administrateur peut rapidement et facilement créer un exercice avec tous les composants de base à sa disposition et ainsi étoffer le catalogue d'exercices. De plus, un générateur automatique a été développé afin d'être une source inépuisable de questions.

Notre implémentation de l'extension est open source et se trouve sur le dépôt Github de INGInious:

`https://github.com/UCL-INGI/INGInious-problems-electrical`

Références

[1] Circuitlab. https://www.circuitlab.com/editor/#?id=7pq5wm, 2019.

[2] Dcaclab. https://dcaclab.com/fr/lab?from_main_page=true, 2019.

[3] Stepik. https://stepik.org/lesson/9184/step/1?unit=1734, 2019.

[4] Mastering physics. https://www.pearsonmylabandmastering.com/northamerica/masteringphysics/, 2018.

[5] What is inginious. https://inginious.readthedocs.io/en/v0.5/teacher_doc/what_is_inginious.html, 2019.

[6] Understand inginious. https://inginious.readthedocs.io/en/v0.5/dev_doc/understand_inginious.html, 2019.

[7] Plugin inginious. https://inginious.readthedocs.io/en/v0.5/dev_doc/plugins.html, 2019.

[8] mxgraph tutorial. https://jgraph.github.io/mxgraph/docs/tutorial.html, 2018-10.

[9] Exemples mxgraph. http://jgraph.github.io/mxgraph/javascript/index.html, 2018.

[10] mxconstants. https://jgraph.github.io/mxgraph/docs/js-api/files/util/mxConstants-js.html, 2019.

[11] Paolo Nenzi Holger Vogt, Marcel Hendrix. Ngspice users manual version 28 plus (describes ngspice master branch version). 2018.

[12] Spice quriks. https://www.allaboutcircuits.com/textbook/reference/chpt-7/spice-quirks/, 2019.

[13] Michel Verleysen Vincent Wertz Kouider Ben-Naoum, Olivier Pereira. *Mathématiques 1 Théorie*. SICI, ciaco, 2013.

[14] Loi de kirchhoff. http://www.elektronique.fr/cours/lois-de-kirchhoff.php, 2019.

[15] J.David Irwin et R. Mark Nelms. *Engineering Circuit Analysis*. John Wiley, 2015.

[16] Olivier Bonaventure. *Réseaux Informatiques - Notes de cours*. Service d'Impression du Cercle Industriel asbl, 2016.

[17] Mobile and embedded computing. https://moodleucl.uclouvain.be/course/view.php?id=12489, 2019.

[18] Kevin Wayne Robert Sedgewick. *Algorithms*. Addison-Weslay, 2011.

[19] Joshua Cantrell. Writing simple spice netlists. 2018.

[20] Inginious github. https://github.com/UCL-INGI/INGInious, 2018.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**