Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études
pour l'obtention du diplôme d'un Master en Informatique

Thème

Recherche efficace des compositions skylines des services web

Réalisé par :

- YAZIT Ibrahim

- OUTADJER Wafaa

Présenté le 04 Juillet 2017 devant la commission d'examinassion composée de :

- ***F.Hadjila***	***(Encadreur)***
- ***I.Smahi***	***(Président)***
- ***M.Merzoug***	***(Examinateur)***

Année universitaire 2016/2017

Remerciements

Tout d'abord, nous tenons à remercier Allah, Le Tout Puissant, de nous avoir donné la santé, la volonté et la patience pour mener à terme notre formation de Master.

Nous tenons à exprimer nos profonds remerciements à messieurs les membres du jury

Nous Remercions :

Mr. HADJILA Fethallah qui nous a fourni le sujet de ce mémoire et nous a guidés de ses précieux conseils et suggestions, et la confiance qu'il nous a témoignés tout au long de ce travail.

Mr Merzoug M. et Mr Smahi I. qu'ils veuillent trouver ici l'expression de nos profonds respects pour avoir eu l'amabilité de vouloir bien faire partie du jury de notre mémoire.

Nous adressons aussi nos remerciements à Mr BENAMMAR chef de département de l'Informatique et à tous les enseignants de la filière Informatique.

A toutes ces personnes, on adresse nos sincères sentiments de gratitude et de reconnaissance.

Dédicace

A mes chers parents Ahmed et Malika.

Pour leur amour, leur patience, leur soutien et leurs encouragements.

Que ce travail soit le témoignage de ma plus profonde affection et de ma reconnaissance.

A mes frères Mohamed et Youcef.

A mes chers oncles et tantes Abdel Ghani, Mohamed, Farid, Omar, Fatima, Sakina, Zoubida, Hafida, Souad.

A toute la famille.

A tous mes amis

Abderrahim KHARBOUCHE, Ayoub BENSIDHOUM, Ayoub BILEM, Ouassini FASLA, Radia BENDIMERAD, Youcef HAMEL, Youcef MOUALEK, Wafaa OUTADJER, Walid SAKHI.

A tous mes camarades de la promotion LMD/MI 2012/2017

Je dédie ce mémoire.

Ibrahim

Dédicace

*A*u nom d'Allah, le Tout Miséricordieux, le Très Miséricordieux Tout d'abord je tiens à remercier le tout puissant de m'avoir donné le courage et la patience pour arriver à ce stade afin de réaliser ce travail que je dédie :

*T*out d'abord à mes parents qui m'ont apporté tout le confort et les moyens pour avoir le meilleur parcours possible tout en me soutenant tout au long de mon cursus universitaire en me poussant à donner le maximum de moi-même. Que Dieu les garde pour moi.

A mon cher frère Younes.

A ma chère sœur samia et son mari et ses enfants Sondos, Islem et Sirine.

A toute ma grande famille : Outadjer et Souier.

A mon binôme Ibrahim que j'estime beaucoup.

*J*e le dédié à mes amies les plus proches : Imen, Souad, Imen, Meriem et Yasmine …

A tous les amis de l'université d'Abou Bekr Belkaïd, à mes camarades de 2 ème année master MID promotion 2017.

A tous ceux que j'aime. Je dédie ce modeste travail.

Wafaa

Table des matières

Chapitre 2 : Approches proposées

Introduction générale

Introduction générale

I. Contexte

L'un des objectifs fondamentaux de l'informatique est de permettre l'interopérabilité entre différents logiciels et applications de données fonctionnant sur une variété de plates-formes. L'introduction des services Web a été essentielle pour le changement de paradigme dans les structures commerciales, ce qui leur permet d'externaliser les fonctionnalités requises des fournisseurs Web tiers grâce à la composition du service.

II. Problématique

Le nombre croissant de services Web générera un nombre important de services Web avec des fonctionnalités similaires. Cela pose également un nouveau défi pour la composition du service : sélectionner les fournisseurs de services appropriés qui obtiennent une composition avec la meilleure réponse du la Qualité du service Web (QoWS) souhaité par l'utilisateur, ce genre de composition est appelé « Composition Skyline ».

Un certain nombre d'approches de sélection de services ont été développées qui dépendent du calcul d'une fonction objective prédéfinie. Un mécanisme de pondération est utilisé lorsque les utilisateurs expriment leur préférence par rapport à des paramètres de qualité différents (et parfois contradictoires) en tant que poids numérique. La composition obtenant la valeur la plus élevée à partir de la fonction objective sera sélectionnée et renvoyée à l'utilisateur.

Il existe deux grandes limites à ces approches. Tout d'abord, il a une tâche assez exigeante pour les utilisateurs de transformer leurs préférences personnelles en poids numériques. Sans une connaissance détaillée de la QoWS de toutes les compositions possibles, les utilisateurs peuvent ne pas être en mesure de prendre une décision de compromis précise entre différents aspects de la qualité à l'aide de nombres. Deuxièmement, chaque fois que les poids sont modifiés, une recherche complètement nouvelle doit être effectuée. Une recherche exhaustive serait coûteuse du point de vue informatique car les compositions possibles augmenteront exponentiellement avec le nombre de services impliqués.

Le but de notre travail est de répondre à la problématique d'accélération de la recherche des skylines composés. En effet certains algorithmes existants sont peu efficaces en termes de temps d'exécution, et il est toujours préférable de mettre en œuvre des approches plus rapide, plus optimale et surtout elle répond aux besoins des clients.

III. Contribution

Dans le cadre de ce mémoire, nous proposons deux heuristiques la première se base sur une fonction objective et la deuxième est basée sur une seule passe de la collection de service, dans les deux cas nous sélectionnons les k skylines de chaque service, ensuite nous appliquons ces meme heuristiques pour obtenir les compositions skylines en un délai limité.

IV. Plan du mémoire

Le reste de la mémoire est organisé comme suit :

- **Le premier chapitre :** est consacré à définir le problème composition de service avec QoS et comment améliorer la performance de la recherche des compositions de service tous en un temps raisonnable.
- **Le deuxième chapitre :** introduit les différentes heuristiques utilisées, et présenter aussi le prototype et les résultats obtenus.
- **La conclusion générale :** résume les résultats de notre travail, et présenter les perspectives que nous souhaitons réaliser dans le futur.

Chapitre I

Sélection des services web

I. Introduction

De nos jours les différentes organisations et entreprises s'orientent vers des architectures à base des services web pour le développement et l'intégration des systèmes d'information ou d'applications.

Les services web sont un atout idéal pour l'interopérabilité et l'intégration des applications sur internet car ils ne dépendent pas des langages, des environnements sous-jacents ou des systèmes d'exploitation.

Dans ce chapitre, nous allons définir ce qu'un service web et aussi présenter les différentes technologies utilisées par les services web, nous allons aussi définir la problématique de composition des services, ensuite nous parlerons des méthodes de sélections des services web afin de proposer quelque solution pour ce problème et pour finir nous allons parler des différents domaines d'utilisation des services web.

II. Les services web

II.1. Définition

Les Services Web sont, en quelque sorte, le prolongement de la programmation objet. Ainsi, un Web Service est donc une sorte d'objet avec une seule fonctionnalité permettant, avec d'autres Web Services, la composition d'une application plus large pouvant avoir plusieurs fonctionnalités. Il est aussi une unité logique applicative accessible en utilisant les protocoles standards d'Internet.

Les services web émergent au début des années 2000 dans le monde des systèmes d'information dans le contexte de la mise en œuvre d'Architectures Orientées Services (SOA : Service Oriented Architecture), dont l'objectif est de rendre plus modulaire et moins propriétaire le développement des logiciels des applications informatiques, en implémentant les fonctions applicatives élémentaires sous forme de modules.

Les services web sont des applications qui relient des programmes, des objets, des bases de données ou des processus d'affaires à l'aide de XML et de protocoles internet standard. [10]

II.2. Les technologies de base des services web

Internet est devenu le moyen de communication, centaines de millions d'utilisateurs qui échangent des informations numériques à travers des serveurs. Ce succès d'Internet s'est construit autour des technologies suivantes : TCP/IP, HTTP, HTML, WSDL, REST, XML-RPC, SOAP et UDDI.

- **TCP/IP :** (Transmission Control Protocol) / (Internet Protocol) est utilisé pour la connectivité.
- **HTML :** (Hyper Text Markup Language) le langage de présentation des données.
- **XML :** (eXtensible Markup Language) est un métalangage basé sur le balisage extensible.
- **REST :** (Representational State Transfer) est une architecture des services web.
- **XML-RPC :** Un protocole qui utilise XML comme langage pour créer des messages RPC.
- **XML Schéma :** est un langage de description de format du document XML qui a été mise au point par W3C permettant de définir une la structure d'un document XML.
- **SOAP :** (Simple Object Access Protocol) est un protocole de la famille XML servant à l'échange d'informations dans un environnement distribué et décentralisé. Il est considéré comme la technologie la plus importante des services web. [7]
- **WSDL :** (Web Services Description Language) est un langage utilisé pour décrire et publier le format et les protocoles d'un service web en utilisant le langage XML.
- **UDDI :** (Universal Description, Discovery and Integration) C'est un format de schéma XML permettant de décrire un service Web en précisant les méthodes disponibles, les formats des messages d'entrée et de sortie et comment y accéder. [9]

III. Problématique

La sélection des compositions de services « QoS-aware service composition », est l'une des problématiques les plus importantes de l'architecture orientée service, dans ce qui suit nous allons monter comment fonctionne cette dernière en illustrant ça avec des exemples.

III.1. Préliminaires

Dans cette section nous définirons quelques notions que nous allons les utiliser par la suite dans un exemple pour motiver les idées clés.

- **Relation de dominance :**

Soit p et q deux vecteurs multi dimensionnel, définis sur l'espace de dimensions D.

p domine q sur D, noté par p < D q, si p est strictement préféré ou égal à q sur toutes les dimensions de D et p est préféré à q sur au moins une dimension ∀ di∈ D, p (di) ≤ di q (di) ⋀∃di ∈D, p (di) < di q (di). Pour plus de lisibilité, p < D q sera simplement noté p < q. [3]

- **Qualité de service web : (QoS)**

Elle représente des critères non fonctionnels qui permettent d'évaluer le comportement d'une application ou un système. Exemples : gigue, débit…

- **Skyline :**

E : c'est un ensemble de points multidimensionnels définis sur D.

P : un sous ensemble d'axes de D.

Le Skyline de l'ensemble de données E sur l'espace de dimensions D, avec P l'ensemble des préférences associées à D, est l'ensemble des points qui ne sont dominés par aucun autre point dans E :

Sky (D, E) P = {p ∈ E| (∀q ∈ E, ¬ (q ≺D p))}. Si P =∅, Sky (D, E) P = E. [3]

- **Score :**

Le score d'un SEP (plan d'exécution séquentiel) qui signifie une suite d'opérations de service web.

Il est calculé comme étant $\Sigma_j^k \sum_i^n q_i\ (op_i)$ où n est le nombre d'opérations dans le SEPx, qj est un attribut QoWS donné dans le Tableau 3 et k est le nombre d'attributs QoWS. Le score d'une CSEPx est la somme des scores de ses SEPx.

Par exemple, nous avons un score (SEPA) = 1,5 + 0,8 + 4 = 6,3 alors que le score (CSEP (A, F)) = score (SEPA) + score (SEPF) = 6,3 + 8,8 = 15,1. Sur la base de la définition du score et en supposant que les valeurs de qualité inférieure sont toujours préférées 1, nous avons l'observation suivante.

III.2. Formalisation de problème

Pour plus de clarté, nous utilisons le terme SEPx pour se référer spécifiquement au plan d'exécution de service pour un seul service. Le S-Sky est utilisé pour désigner le skyline calculé à partir des SEPx.

Nous utilisons le terme CSEPx pour désigner le plan d'exécution du service pour un service composite. Un autre concept important utilisé dans notre approche est le nombre de SEPx et de CSEPx.

Une note est définie en fonction de la QoWS d'un SEPx (ou d'un CSEPx). Rappelons que la QoWS d'un SEPx (ou d'un CSEPx) est calculée en agrégant celles de ses opérations de service aux membres.

Nous nous concentrons sur les fonctions d'agrégation de type qui peuvent exprimer la QoWS d'un SEPx (ou d'un CSEPx) comme la somme de ses QoWS d'opérations.

Pour certaines fonctions d'agrégation qui sont généralement définies comme le produit des attributs QoWS des opérations membres (p. Ex., Fiabilité et disponibilité), nous prenons un logarithme pour effectuer une adaptation comme le montre le tableau I.1.

Pour une fonction moyenne (par exemple agrégation de Réputation), nous utilisons sa version agrégée.

Paramètre de QoWS	Fonction d'agrégation
Latence	$\Sigma_{i=1}^{n} latence(op_i)$
Fiabilité	$\Sigma_{i=1}^{n} \log(fiabilité(op_i))$
Disponibilité	$\Sigma_{i=1}^{n} \log(disponibilité(op_i))$
Cout	$\Sigma_{i=1}^{n} frais(op_i)$
Réputation	$\Sigma_{i=1}^{n} réputation(op_i)$

Tableau I.1. Fonction d'agrégation

III.3. Exemples

- **Exemple 1 : (service composé)**

Envisager le développement d'un service Web composite, TravelAssistant, qui fournit des services d'assistance aux voyageurs. Les services Web typiques auxquels il faudrait avoir accès aux différents services : TripPlanner, Map et Weather.

« TripPlanner » fournit des informations de voyage de base, telles que les compagnies aériennes, les hôtels et les attractions locales. Autre que cela, les utilisateurs peuvent également être intéressés à consulter la carte de la ville et les transports locaux en accédant au service « Map ».

La condition météorologique pendant les jours de voyage est un facteur important qui rend le service « Weather » pertinent.

Le développeur peut faire face à un certain nombre d'options pour chacun de ces services car il y a plusieurs fournisseurs de logiciels concurrents pour offrir des fonctionnalités similaires.

Par exemple, le tableau I.2 montre les cinq fournisseurs de services de carte possibles et quatre fournisseurs de planificateur de voyage possibles. L'accès à un service Web inclut généralement l'appel d'un ensemble d'opérations. Par exemple, l'accès à un service Map nécessite d'appeler deux opérations : Geocode et GetMap.

Il peut y avoir des contraintes de dépendance entre ces opérations (par exemple, GetMap dépend du géocode). Ainsi, ces opérations peuvent être agencées en une séquence par rapport aux contraintes de dépendance, appelée Plan d'exécution de service (SEPx).

La QoWS d'un SEPx est calculée en agrégeant celles de ses opérations de service membres en utilisant un ensemble de fonctions d'agrégation prédéfinies.

Exemple	Opération	Latence	Cout	Réputation
Fournisseurs de cartes				
A	GeoCode	0.4	0.7	3
	GetMap	0.9	0	3
B	GeoCode	0.6	0.4	3
	GetMap	1.9	0.4	3
C	GeoCode	0.4	0.2	2
	GetMap	1.5	0.9	2
D	GeoCode	0.2	0.7	3
	GetMap	0.9	0.4	3
E	GeoCode	0.5	0.7	2
	GetMap	0.7	0.5	2
Planificateurs de voyage				
F	SearchTrip	1	0.2	3
	GetTrip	2	0.8	3
G	SearchTrip	1	0.1	3
	GetTrip	1	1	1
H	SearchTrip	3	1.1	2
	GetTrip	2	1	2
I	SearchTrip	2	1	2
	GetTrip	2	0.7	2

Tableau I.2. Fournisseurs de cartes et planificateurs de voyage

- **Exemple 2 : (Dominance)**

Considérez trois paramètres de QoWS, la latence, le cout et la réputation, qui capturent le temps de réponse, le coût monétaire et la cote des utilisateurs d'un service. Supposons qu'une notation prenne des valeurs compris entre [1, 5] et qu'une valeur plus petite reflète une meilleure cote.

La latence et le cout d'une SEPx sont calculés comme la somme de ceux de ses opérations membres. La réputation est établie comme la moyenne de ceux des opérations membres.

La QoWS des SEPx pour les cinq fournisseurs de cartes possibles est (1.3, 0.7, 3), (2.5, 0.8, 3), (1.9, 1.1, 2), (1.1, 1.1, 3) et (1.2, 1.2, 2).

Pour deux fournisseurs p1 et p2, si SEP1 est aussi bon que SEP2 dans tous les aspects QoWS et mieux que SEP2 dans au moins un aspect QoWS, alors SEP1 domine SEP2, ou

noté SEP1 ≺ SEP2. Par conséquent, SEPA domine SEPB et SEPC, et SEPD domine SEPE.

Étant donné que le SEPA et SEPD ne sont pas dominés par d'autres fournisseurs, il est dit qu'ils sont considérés comme des skylines du premier service.

D'autre part, puisque SEPB, SEPC et SEPE sont dominés par SEPA et SEPD, respectivement, ils ne sont pas dans l'ensemble des skylines.

Plus formellement, un skyline de service ou S-Sky peut être considéré comme un ensemble de SEPs qui ne sont pas dominés par d'autres SEPs.

Les fournisseurs de services dans le skyline représentent les meilleurs compromis entre les différents aspects de qualité intéressés par l'utilisateur. Autre exemple, le SEPF et le SEPG constituent le skyline du service Trip-Planner comme le montre le Tableau I.2.

- **Exemple 3 : (Skyline composé)**

Considérons la composition (appelée TravelAssistance) de deux services : TripPlanner et Map. Un plan d'exécution de service composite ou CSEPx qui se compose de quatre opérations (TripSearch, GetTrip, Geocode, GetMap) peut être généré pour accéder au service composite. La QoWS d'une CSEPx est calculée en agrégeant la QoWS des SEPx d'une manière semblable à celle décrite dans l'exemple 2.

Un skyline de service composite ou C-Sky est un ensemble de CSEPs qui ne sont pas dominés par d'autres CSEPs. Une façon naïve de trouver le C-Sky de la composition de TravelAssistance est de générer les 20 CSEPx possibles et de vérifier ensuite la dominance du service entre eux.

- **Exemple 4 : (Calcul des Skylines composés)**

Pour une composition de service avec m classes et en supposant qu'il existe n1, ..., nm fournisseurs pour chacun des m classes, $\pi_{i=1}^{m} n_i$ nombre de compositions doivent être considérées afin de trouver le C-Sky. Pour une composition complexe avec un m relativement grand (par exemple, 10), un nombre modéré de fournisseurs pour chaque service (par exemple, 100) introduira des frais généraux prohibitifs (par exemple, 10010 compositions doivent être considérées). Une solution intuitive qui peut réduire considérablement le temps d'exécution est de calculer le C-Sky à partir de skylines service individuel. Ceci est dû à une observation clé : un C-Sky peut être complètement déterminé en ne considérant que les SEPx de S-Skies. Dans ce cas, seul N = $\pi_{i=1}^{m} k_i$ où ki nombre maximal de compositions doit être évalué, où ki est la taille du ième S-Sky.

Comme la taille d'un S-Sky est généralement beaucoup plus petite que le nombre de fournisseurs, le coût de calcul peut être réduit avec plusieurs ordres de grandeur. Un exemple est donné dans le tableau I.3. Au lieu de considérer les 20 CSEPx, le C-Sky peut être déterminé par seulement quatre CSEPx candidats, qui sont formés en combinant les skylines des deux services qui composent le service TripPlanner.

CSEP	Latence	Cout	Réputation
CSEP(A, F)	4.3	1.7	3
CSEP(A, G)	3.3	1.8	2.5
CSEP(D, F)	4.1	2.1	3
CSEP(D, G)	3.1	2.2	2.5

Tableau I.3. Exemple de C-Sky

IV. Méthodes de sélections

La sélection de services web, a fait l'objet de plusieurs travaux, de façon générale on distingue 03 grandes classes (voir la figure I.1. [4]. [5]) :

La sélection mono objective, La sélection multi objective, La sélection hybride (mono et multi objective).

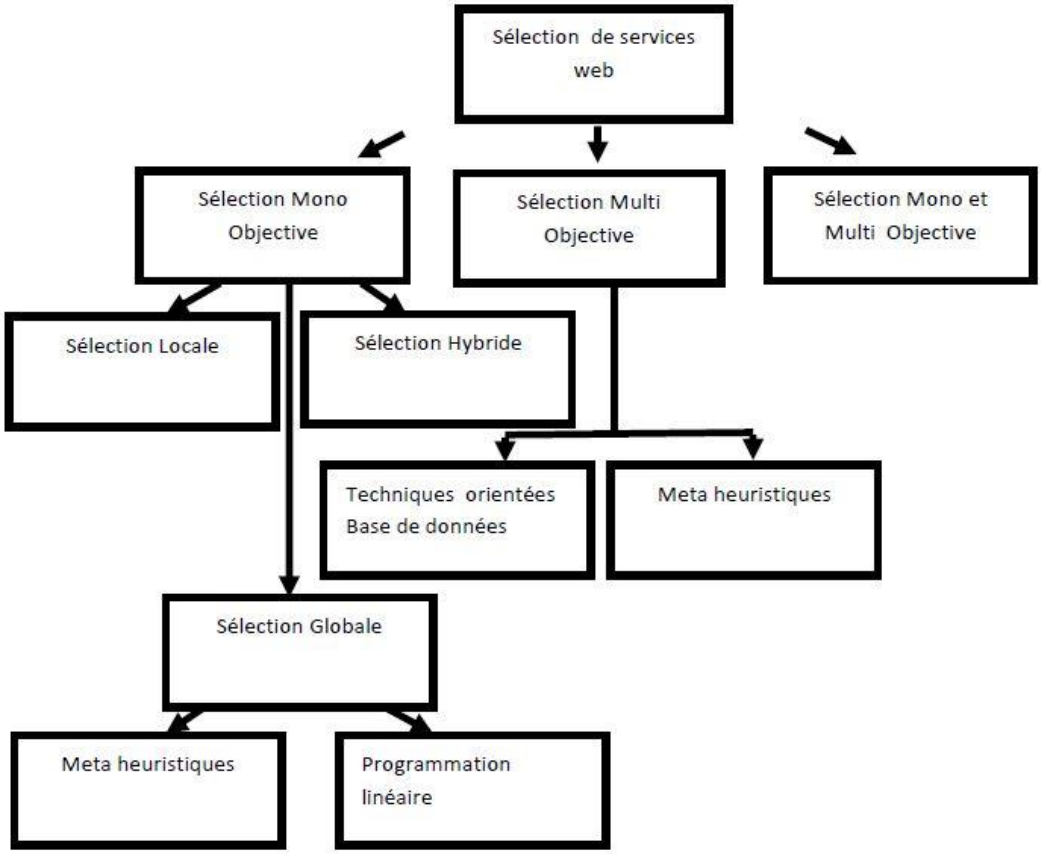


Figure I.1. Les différentes classes de sélection de services

IV.1. La sélection mono objective

Ce type de sélection suppose que les k valeurs de QOS sont agrégés ont un seul score, en utilisant une fonction objective qui donne un poids pour chaque attribut de service. Dans cette catégorie il y a trois sous classes :

La sélection locale, globale et hybride (globale et locale).

- **Sélection globale :**

L'approche global explore un espace de recherche dont les nœuds sont des compositions complètes (c.à.d. contenant toutes les tâches), elle est devisée en deux sous classes : exacte et approximative.

- **Optimisation exacte :**

Basée sur la programmation entière ou la programmation par contrainte, ou les énumérations exhaustives, les résultats de ces méthodes sont optimaux mais le temps d'exécution est exponentiel.

- **Optimisation approximative :**

Cette catégorie utilise des heuristiques ou des méta heuristiques et consiste à explorer une partie de l'espace de recherche.

- **Sélection Locale**

Elle consiste à choisir un seul service de chaque classe en utilisant une fonction objective (une fonction pour chaque service), ensuite faire les n compositions possibles en se basant sur les résultats qui correspondent aux n classes.

Cette approche possède une complexité linéaire O(l) [1], en plus elle est fortement adaptée aux environnements distribués, en effet la gestion de la QOS (mesures, mise à jour … est faite par des facilitateurs distribués).

- **Sélection Hybride**

Les méthodes de sélections globales et exactes sont fortement liées aux nombres de services, puisqu'une variable binaire est associée à chaque service (n*k variables sachant que n est le nombre de classes, et k est le nombre de services par classe). D'où nous comprenons le caractère exponentiel et la non scalabilité de ces approches.

Pour remédier à ce problème, nous devons passer par deux phases d'optimisation, une phase globale qui considère des intervalles de valeurs de QoS comme variables à la place des services directs et une deuxième phase où nous utilisons une heuristique.

IV.2. Sélection Multi-Objectives

La sélection multi objectif est une branche de l'optimisation combinatoire dont la particularité est de chercher à optimiser simultanément plusieurs objectifs d'un même problème (contre un seul objectif pour l'optimisation combinatoire classique). Elle se distingue de l'optimisation multidisciplinaire par le fait que les objectifs à optimiser portent ici sur un seul problème.

Les problèmes multi objectifs ont un intérêt grandissant dans l'industrie. Où les responsables sont contraints de tenter d'optimiser des objectifs contradictoires. Ces problèmes peuvent être NP-Hard même si la version mono-objective ne l'est pas (c'est le cas du problème d'affectation par exemple). Leur résolution en des temps raisonnables devient nécessaire et alimente une partie des chercheurs travaillant dans la recherche opérationnelle.

IV.3. Sélection Mono et Multi-objective

Cette catégorie se base sur les fonctions de la sélection mono-objectif et aussi les fonctions de la sélection multi-objectif.

V. Domaines d'applications

Les services web sont très utilisés de nous jour et dans différents domaines, dans cette section nous allons parler de quelques domaines d'utilisation des services web.

V.1. Cloud Computing

Le cloud computing, ou l'informatique en nuage ou nuagique, est l'exploitation de la puissance de calcul ou de stockage de serveur informatiques distants par l'intermédiaire d'un réseau, généralement internet.

Ces serveurs sont loués à la demande, le plus souvent par tranche d'utilisation selon des critères techniques (puissance, bande passante, etc.) mais également au forfait.

Le cloud computing se caractérise par sa grande souplesse : selon le niveau de compétence de l'utilisateur client, il est possible de gérer soi-même son serveur ou de se contenter d'utiliser des applicatifs distants.

V.2. E Gouvernement (Administration électronique)

L'administration électronique peut se développer dans tout type d'administration ou de service public, en contact avec le public (front-office) ou non (back-office). Elle se caractérise par l'emploi de technologie de l'information et la communication visant à améliorer les processus, la communication entre usagers et administrations ou entre administrations et l'efficacité de l'administration, que ce soit sur le plan des délais, de la qualité, ou de la productivité des agents publics.

Les supports de l'administration électronique sont nombreux. On pense souvent d'abord à Internet (services web sur ordinateur ou téléphone mobile), mais un projet d'administration électronique peut aussi s'appuyer sur toute forme de télématique, la communication en champ proche, Bluetooth ainsi que les projets de carte à puce, éventuellement combiné à la biométrie, ainsi que des procédures de vote électronique, ou encore la vidéosurveillance, laquelle peut converger avec l'informatique, la constitution de bases de données et des procédés biométriques de reconnaissance faciale.

V.3. Internet des objets (IoT)

C'est une infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution.

L'IoT est donc « un réseau de réseaux qui permet, via des systèmes d'identification électronique normalisés et unifiés, et des dispositifs mobiles sans fil, d'identifier directement et sans ambiguïté des entités numériques et des objets physiques et ainsi de pouvoir récupérer, stocker, transférer et traiter, sans discontinuité entre les mondes physiques et virtuels, les données s'y rattachant.

VI. Conclusion

Nous avons présenté dans ce chapitre, la problématique de sélection de services web à base de QOS.

Nous avons aussi mis en évidence, les différentes méthodes de sélections des services web, et plus précisément, nous avons étudié la sélection mono objective avec toutes ses variantes, la sélection multi objective, et la sélection hybride.

Nous avons cité quelque domaine d'application des services web et leurs utilités.

Dans le chapitre suivant, nous présentons 03 approches qui vont nous permettre de sélectionner des services à base de QOS.


Chapitre II
Approches proposées

I. Introduction

Les Requêtes Skyline ont attiré une attention considérable en raison de son importance dans de nombreuses applications telles que la prise multicritère de décision, l'extraction de données, et des requêtes de préférence [1] ; ils renvoient un ensemble d'objets de données intéressantes de tous les autres objets. Et pour ce dernier, nous considérons 03 algorithmes « algorithme à base de la somme de la QoS, algorithme naïve, algorithme one pass » qui se basent sur la minimisation des objets.

II. Outils de développement

Avant de commencer l'implémentation de notre application, nous allons tout d'abord spécifier les outils utilisés qui nous ont semblé être un bon choix vu les avantages qu'ils offrent.

II.1. JAVA

L'application est développée en utilisant le langage de programmation Java version 1.6. Java est un langage de programmation orienté objet et familier. Créer par James Goslinget Patrick Naughton présenté officiellement le 23 mai 1995 au Sun Word.Sa simplicité, sa robustesse, sa portabilité ainsi que sa performance lui ont permis d'être le choix préféré pour le développement de notre application.

II.2. Netbeans

Nous avons écrit notre application en NetBeans version 6.8. NetBeans est l'environnement de développent intégré (IDE), placé en open source par Sun en juin 2000 sous licence CDDL etGPLv2 (Common Development and Distribution License). En plus de Java NetBeans permet également de supporter différents autres langages comme python, C, C++, Java Script, XML, PHP, HTML. Il comprend toutes les caractéristiques d'un IDE moderne. Et le concurrent direct d'Eclipse sur le marché des environnements open source. En tirant avantage de cette trousse à l'outil gratuit, basée sur des standards, les développeurs, peuvent concevoir des applications complexes plus rapidement, avec une grande assurance de robustesse et de concevoir des applications qui résisteront à l'épreuve du temps. Pour cela le choix de NetBeansétait fondamental.

II.3. JDOM

JDOM (acronyme de l'anglais Java Document Object Model), est une bibliothèque open source pour manipulation des fichiers XML (eXtensible Markup Language) en Java. JDOM permet aussi de vérifier que les données contenues dans les éléments respectent la norme XML. Elle intègre DOM et SAX, et supporte XPath et XSLT.

Utilise des analyses syntaxiques externes pour construire les documents. La dernière version 2.0.4 le 8 novembre 2012.

III. Contribution

Dans cette section nous allons présenter les différents algorithmes que nous avons développés dans notre application.

III.1. Algorithme à base des requêtes topK

III.1.1. Algorithme

```
1.  Entrée : data[][][]
2.  Sortie : Tableau C-TopL
3.  Début
4.  Pour i=1 jusqu'à n+1
5.    Pour j=1 jusqu'à d+1
6.      Pour k=1 jusqu'à p+1
7.            Générer_Base_Aléatoire (data [i] [j] [k])
8.      Fin pour
9.    Fin pour
10. Fin pour
11. Pour i=1 jusqu'à n+1
12.   Pour j=1 jusqu'à d+1
13.     Pour k=1 jusqu'à p+1
14.           Aggregation (data [1][1][1] +… + data[1][1][p])
15.     Fin pour
16.   Fin pour
17. Fin pour
18. Pour i=1 jusqu'à n+1
19.   Pour j=1 jusqu'à d+1
20.     Pour k=1 jusqu'à p+1
21.           Trier (data [i][j][k])
22.     Fin pour
23.   Fin pour
24. Fin pour
25. Entier nb_topk // le nombre topk que nous souhaitons conservés
26. Pour i=1 jusqu'à n+1
27.   Pour j=1 jusqu'à nb_topk+1
28.     Pour k=1 jusqu'à p+1
```

29. TopL-Classe$_i$ [i][j][k] = data [i] [j][k]
30. Fin pour
31. Fin pour
32. Fin pour
33. Pour i=1 jusqu'à n+1
34. Pour j=1 jusqu'à nb_sk+1
35. Pour k=1 jusqu'à p+1
36. Candidat_ TopL [i][j][k] = TopL-Classe$_i$[i][j][k]+ TopL-Classe$_{i+1}$[i][j][k]+ … + TopL-Classe$_n$ [i][j][k]
37. Fin pour
38. Fin pour
39. Fin pour
40. Pour i=1 jusqu'à n+1
41. Pour j=1 jusqu'à nb_sk+1
42. Pour k=1 jusqu'à p+1
43. C-TopL[i][j][k] = Trier (Candidat_ TopL)
44. Fin pour
45. Fin pour
46. Fin pour
47. Afficher(TopComposition)
48. Fin

III.1.2. Explication

- **Entrée :**

Un tableau de QoS pour chaque classe noté tabn n est le numéro de la classe.

- **Etape 1 :** Calculer la somme des QoS pour chaque instance des n classes, et les stockées dans un vecteur noté TopL-Classen.
- **Etape 2 :** Tri le résultat (TopL-Classen) par ordre croissant.
- **Etape 3 :** Entrer le nombre de TopK que vous souhaitez garder. Noté nb_topk.
- **Etape 4 :** Faire toutes les combinaisons possibles entre les TopL-Classe choisis, et les stockés dans un vecteur noté C-TopL.

- **Sortie :**

Liste TopComposition.

III.1.3. Exemple

Nous considérons 03 classes, chaque classe à 04 instances et chaque instance à 03 critères.

Nous supposons que chaque service est constitué d'une seule opération.

	Cr_1	Cr_2	Cr_3
$Instance_1$	0.8	0.5	0.2
$Instance_2$	0.3	0.4	0.2
$Instance_3$	0.5	0.6	0.3
$Instance_4$	0.6	0.5	0.4

Tableau II.1. Tableau des critères pour classe 1

	Cr_1	Cr_2	Cr_3
$Instance'_1$	0.3	0.5	0.4
$Instance'_2$	0.7	0.2	0.6
$Instance'_3$	0.4	0.8	0.4
$Instance'_4$	0.1	0.2	0.5

Tableau II.2. Tableau des critères pour la classe 2

	Cr_1	Cr_2	Cr_3
$Instance''_1$	0.6	0.2	0.5
$Instance''_2$	0.7	0.5	0.2
$Instance''_3$	0.5	0.3	0.4
$Instance''_4$	0.3	0.3	0.6

Tableau II.3. Tableau des critères pour la classe 3

- **Etape 1 :**

Calcul des sommes pour chaque instance :

$\sum Cri = Cri + Cri+1 + \ldots + Crp$.

- Instance 1 : $\sum Cri = 0.8+0.5+0.2 = 1.5$
- Instance 2 : $\sum Cri = 0.3+0.4+0.2 = 0.9$
- Instance 3 : $\sum Cri = 0.5+0.6+0.3 = 1.4$
- Instance 4 : $\sum Cri = 0.6+0.5+0.4 = 1.5$

TopL-Classe1 = {(instance1, 1.5), (instance2, 0.9), (instance3, 1.4), (instance4, 1.5)}

Refaire le même travail pour les deux autres tableaux.

TopL-Classe2 = {(instance'1, 1.2), (instance'2, 0.5), (instance'3, 1.6),(instance'4, 0.8)}

TopL-Classe3 = {(instance’’1,1.3),(instance’’2, 1.4),(instance’’3, 1.2),(instance’’4,1.2)}

- **Etape 2 :**

Trier les différents TopL-Classei.

TopL-Classe1 = {(instance1,1.5),(instance2,1.5),(instance3,1.4),(instance4,0.9)}

Refaire le même travail pour les deux autres tableaux.

TopL-Classe2 = {(instance’1,1.6),(instance’2,1.2),(instance’3,0.8),(instance’4,0.5)}

TopL-Classe3 = {(instance’’1,1.4),(instance’’2,1.3),(instance’’3,1.2),(instance’’4,1.2)}

- **Etape 3 :**

Donnez le nombre de TopK à garder.

Nb_topK=2.

Affichez les TopK à garder.

TopL-Classe1 = {(instance1, 1.5), (instance2, 1.5)}

TopL-Classe2 = {(instance’1, 1.6), (instance’2, 1.2)}

TopL-Classe3 = {(instance’’1, 1.4), (instance’’2, 1.3)}

- **Etape 4 :**

Faire les différentes combinaisons possibles.

Score (comp[k]) = score(TopL-Classe1 [j1])+ score(TopL-Classe2 [j2])+ …+score(TopL-Classen [jn]) , j1…jn: parcourent toutes les valeurs possibles des instances de services

Score(C-TopL [1]) = ∑ C-TopL-Classe1[1] + C-TopL-Classe2[1] + C-TopL-Classe3[1] =1.5+1.6+1.4=4.5

Score(C-TopL [2]) = ∑ C-TopL-Classe1[1] + C-TopL-Classe2[1] + C-TopL-Classe3[2] =1.5+1.6+1.3=4.4

Score(C-TopL [3]) = ∑ C-TopL-Classe1[1] + C-TopL-Classe2[2] + C-TopL-Classe3[1] =1.5+1.5+1.4=4.4

Score(C-TopL [4]) = ∑ C-TopL-Classe1[1] + C-TopL-Classe2[2] + C-TopL-Classe3[2] =1.5+1.5+1.3=4.3

Score(C-TopL [5]) = ∑ C-TopL-Classe1[2] + C-TopL-Classe2[1] + C-TopL-Classe3[1] =1.5+1.6+1.4=4.5

Score(C-TopL [6]) = ∑ C-TopL-Classe1[2] + C-TopL-Classe2[1] + C-TopL-Classe3[2] =1.5+1.6+1.3=4.4

Score(C-TopL [7]) = ∑ C-TopL-Classe1[2] + C-TopL-Classe2[2] + C-TopL-Classe3[1] =1.5+1.5+1.4=4.4

Score(C-TopL [8]) = ∑ C-TopL-Classe1[2] + C-TopL-Classe2[2] + C-TopL-Classe3[2] =1.5+1.5+1.3=4.3

Trier le tableau des C-TopL et afficher les nb_topK (dans notre cas c'est les deux meilleures) C-TopL.

C-TopL = ((comp[1],4.5), (comp[5],4.5), (comp[2],4.4), (comp[3],4.4), (comp[6],4.4), (comp[7],4.4), (comp[4],4.3), (comp[8],4.3))

TopComposition = {(comp[1],4.5), (comp[5],4.5)}

Donc les meilleurs résultats sont la combinaison entre l'instance1 avec l'instance'1 et l'instance''1, l'instance2 avec l'instance''1 et l'instance''1.

III.2. Algorithme à Une Seule Passe (One Pass Algorithm)

III.2.1. Algorithme

1 : Entrée : m Skylines pour chaque service
2 : Sortie: Tableau C-Sky
3 : N = $\Pi_{i=1}^{m} = |Sk_m|$= ; // Nombre de candidat C-Sky
4 : CSEP1 = Aggregation (SEP11, ..., SEPm1);
5 : C-Sky.add(CSEP1);
6 : pour i=jusqu'à N
7: CSEPi = Enumérer (SK1, ..., SKm) ;
8: est_dominé = Faux ;
9: pour j=1 jusqu'à |C-Sky|
10: CSEPj = C-Sky.get(j) ;
11: si CSEPi.score < CSEPj.score alors
12: si CSEPi ≺ CSEPj alors
13 : C-Sky.enlver(j) ;
14 : fin si
15 : si non
16 : si CSEPj ≺ CSEPi alors
17 : est_dominé =vrai ;
18 : fin si
19 : fin si
20 : fin pour
21 : si est_dominé == False alors
22 : C-Sky.ajouter(CSEPi);
23 : fin si
24 : fin pour

III.2.2. Explication

- **Entrée :**

Un tableau de QoS pour chaque service noté tabn n est le numéro de la classe.

- **Etape 1 :** Calculer les Skylines de chaque service en utilisant la dominance et les stocker dans des tableaux noté Skyn
- **Etape 2 :** Faire les combinaisons possibles entre les tableaux Sky en utilisant une fonction d'agrégation pour les critères.
- **Etape 3 :** Utiliser la dominance pour éliminer tous les C-Sky dominés après l'agrégation des critères.
- **Etape 4 :** Afficher la liste des C-Sky.

- **Sortie :**

Liste des C-Sky.

III.2.3. Exemple

Nous considérons 03 classes, chaque service à 03 instances et chaque instance à 03 critères.

	Cr_1	Cr_2	Cr_3
$Instance_1$	0.5	0.5	0.4
$Instance_2$	0.7	0.2	0.6
$Instance_3$	0.4	0.8	0.4

Tableau II.4. Tableau des critères pour la classe1

	Cr_1	Cr_2	Cr_3
$Instance'_1$	0.2	0.5	0.4
$Instance'_2$	0.7	0.5	0.6
$Instance'_3$	0.6	0.6	0.5

Tableau II.5. Tableau des critères pour la classe 2

	Cr_1	Cr_2	Cr_3
$Instance''_1$	0.1	0.4	0.4
$Instance''_2$	0.6	0.5	0.6
$Instance''_3$	0.8	0.8	0.7

Tableau II.6. Tableau des critères pour la classe 3

- **Etape 1 :**

Nous allons calculer les skylines de chaque classe en utilisant la dominance.

Pour le 1ere classe nous remarquons qu'il n'y a pas de dominance entre les instances donc les 03 instances sont considérées comme des skyline, donc nous les ajoutons au tableau des skylines.

Sky1[1] = (instance1, (0.5,0.5,0.4))

Sky1[2] = (instance2, (0.7,0.2,0.6))

Sky1[3] = (instance3, (0.4,0.8,0.4))

Pour la 2eme classe nous remarquons que l'instance'2 domine l'instance'1, mais entre l'instance'2 et l'instance'3 il n'y a pas de dominant, donc nous ajoutons au tableau des skylines l'instance'2 et l'instance'3.

Sky2[1] = (instance'2, (0.7,0.5,0.6))

Sky2[2] = (instance'3, (0.6,0.6,0.5))

Pour la 3eme classe nous remarquons que l'instance''3 domine l'instance''1 et l'instance''2, donc nous ajoutons au tableau des skylines l'instance''3.

Sky3[1] = (instance''3, (0.8,0.8,0.7))

- **Etape 2 :**

Nous allons faire les différentes combinaisons possibles entre les Sky_n.

score(C-Sky[i])= ((score1(Sky1[j1] +…+score1(Skyn[jn]),…, (scorer(Sky1[j1] +…+scorer(Skyn[jn]),…, (scoreNbreCriteres(Sky1[j1] + scoreNbreCriteres(Skyn[jn]))

J1,j2,…Jn : parcourent toutes les valeurs possibles des instances skylines

r : représente un critere de QoS

C-sky[i]= ((instancej1, instancej2, instancejn), score(C-Sky[i]))

Score (C-sky [1]) = ((instance1, instance'2, instance''3), score(C-Sky[i]))=(0.5+0.7+0.8), (0.5+0.5+0.8), (0.4+0.6+0.7) = ((instance1,instance'2,instance''3), 2.0,1.8,1.7)

Score (C-Sky [2]) = ((instance1, instance'3, instance''3), 1.9,1.9,1.6)

Score (C-Sky [3]) = ((instance2, instance'2, instance''3), 2.2,1.5,1.9)

Score (C-Sky [4]) = ((instance2, instance'3, instance''3), 2.1,1.6,1.8)

Score (C-Sky [5]) = ((instance3, instance'2, instance''3), 1.9,2.3,1.6)

Score (C-Sky [6]) = ((instance3, instance'3, instance''3), 1.8,2.2,1.6)

- **Etape 3 :**

Comparer entre les éléments de C-Sky en utilisant la dominance, nous remarquons que le C-Sky [5] domine C-Sky [2]et C-Sky[6], donc éliminer C-Sky[2] et C-Sky [6] de la liste des C-Sky.

- **Etape 4 :**

Afficher la liste des C-Sky

C-Sky= (((instance1, instance'2, instance''3), 2.0,1.8,1.7), ((instance2, instance'2, instance''3), 2.2,1.5,1.9), ((instance2, instance'3, instance''3), 2.1,1.6,1.8), ((instance3, instance'2, instance''3), 1.9,2.3,1.6))

III.3. Algorithme naïve

III.3.1. Algorithme

```
1.  Entrée : n tableau tabn // n est le nombre de classes
2.  Sortie : Tableau C-Sky
3.  Début
4.  Pour i=1 jusqu'à n+1
5.    Pour j=1 jusqu'à d+1 // d nombre d'instances par classes
6.      Pour k=1 jusqu'à p+1 // p nombre de critères par instances
7.            Générer_Base_Aléatoire (tabi [j] [k])
8.      Fin pour
9.    Fin pour
10. Fin pour
11. CSEP1 = Aggregation (SEP11, ..., SEPm1);
12. C-Sky.add(CSEP1);
13. pour i=jusqu'à N
14. CSEPi = Enumérer (SK1, ..., SKm) ;
15. est_dominé = Faux ;
16. pour j=1 jusqu'à |C-Sky|
17. CSEPj = C-Sky.get(j) ;
18. si CSEPi.score < CSEPj.score alors
19. si CSEPi ≺ CSEPj alors
20. C-Sky.enlver(j) ;
21. fin si
22. si non
23. si CSEPj ≺ CSEPi alors
24. est_dominé =vrai ;
25. fin si
26. fin si
27. fin pour
28. si est_dominé == False alors
29. C-Sky.ajouter(CSEPi);
30. fin si
31. fin pour
32. Afficher(C-Sky)
33. Fin
```

III.3.2. Explication

- **Entrée :**

Un tableau de QoS pour chaque service noté tabn n est le numéro de service.

- **Etape 1 :** Faire les combinaisons possibles entre les tabn.

- **Etape 2 :** Utiliser la dominance pour éliminer les C-Sky dominés.
- **Etape 3 :** Afficher les C-Sky obtenus.

➢ **Sortie :**

La liste des C-Sky.

III.3.3. Exemple

Nous considérons 02 classes, chaque classe à 03 instances et chaque instance à 02 critères.

	Cr1	Cr2
Instance 1	0.6	0.4
Instance 2	0.5	0.7
Instance 3	0.3	0.5

Tableau II.7. Tableau des critères pour la classe 1

	Cr1	Cr2
Instance’ 1	0.2	0.7
Instance’ 2	0.6	0.6
Instance’ 3	0.5	0.3

Tableau II.8. Tableau des critères pour la classe 2

- **Etape 1 :**

Score (C-Sky[k])= ((score1(Vec1[j1] +…+score1(Vecn[jn]),…, (scorer(Vec1[j1] +…+scorer(Vecn[jn]),…,(scoreNbreCriteres(Vec1[j1] +…+scoreNbreCriteres(Vecn[jn]))

J1,j2,…Jn : parcourent toutes les valeurs possibles des instances de services.

r : représente un critere de QoS

Score (C-Sky [1]) = (((Score1(Vec1[1]) + Score1(Vec2[1])), ((Score2(Vec1[2]) + Score2(Vec2[2])) = ((0.6+0.2), (0.4+0.7)) = (0.8, 1.1)

Score (C-Sky [2]) = ((0.6+0.6), (0.4 +0.6)) = (1.2,1.0)

Score (C-Sky [3]) = ((0.6+0.5), (0.4 +0.3)) = (1.1,0.7)

Score (C-Sky [4]) = ((0.5+0.2), (0.7 +0.7)) = (0.7,1.4)

Score (C-Sky [5]) = ((0.5+0.6), (0.7 +0.6)) = (1.1,1.3)

Score (C-Sky [6]) = ((0.5+0.5), (0.7+0.3)) = (1.0,1.0)

Score (C-Sky [7]) = ((0.3+0.2), (0.5 +0.7)) = (0.5,1.2)

Score (C-Sky [8]) = ((0.3+0.6), (0.5 +0.6)) = (0.9,1.1)

Score (C-Sky [9]) = ((0.3+0.5), (0.5 +0.3)) = (0.8,0.8)

- **Etape 2 :**

Utiliser la dominance pour éliminer les éléments dominés.

C-Sky [8] domine C-Sky [1] domine C-Sky [9], C-Sky [2] domine C-Sky [3] et C-Sky [6], C-Sky [5] domine C-Sky [7] et C-Sky [8]

- **Etape 3 :**

Afficher la liste des C-Sky

Score (C-Sky) = [Score (C-Sky [2]) , Score (C-Sky [4]), Score (C-Sky [5])]= [(1.2,1.0), (0.7,1.4), (1.1,1.3)]

IV. Prototype

Dans ce qui suit nous présentons les interfaces graphiques de notre application.

IV.1. Création de l'interface

L'interface principale de notre application est constituée de 03 jLabel la première définie le nombre de classes ou service, la deuxième pour le nombre d'exemples par classe et la troisième est utilisée pour stocker le nombre de critères par exemple.

Elle contient aussi 02 jButton le premier est utilisé pour confirmer les choix de nombre de classes, exemples et critères. Et le deuxième est utilisé pour générer des données aléatoires en utilisant la fonction math.Random().

Nous avons aussi utilisé des jCheckBox pour basculer entre les différents algorithmes développés dans l'application, et aussi un chrono qui démarre à l'exécution de chaque algorithme pour sauvegarder le temps d'exécution.

Elle contient aussi 02 jButton, « à propos » qui nous envoie vers une autre interface ou nous avons défini le thème de notre travail et les auteurs de cette application. Et « quitter » pour disposer de l'application.

Figure II.1. Interface de l'application

IV.2. Exécution

1. Enter le nombre de service, exemple et critères.

Figure II.2. Initialisation des données

2. Confirmez votre choix et générez une base aléatoire à base des choix de l'utilisateur.

```
Nombre de classes : 3
Nombre d'instances par classe est : 5
Nombre de critères est : 2

Exemple de service 1
Instances de l'exemple 1 :   0.7193888   0.52549547
Instances de l'exemple 2 :   0.08987141   0.1202299
Instances de l'exemple 3 :   0.48012698   0.50566316
Instances de l'exemple 4 :   0.39488238   0.09632653
Instances de l'exemple 5 :   0.11428809   0.29518902

Exemple de service 2
Instances de l'exemple 1 :   0.9992014   0.44937325
Instances de l'exemple 2 :   0.24974096   0.008316219
Instances de l'exemple 3 :   0.6016469   0.41250247
Instances de l'exemple 4 :   0.10533428   0.42000288
Instances de l'exemple 5 :   0.9319055   0.4880545

Exemple de service 3
Instances de l'exemple 1 :   0.2213394   0.32379305
Instances de l'exemple 2 :   0.5363123   0.991032
Instances de l'exemple 3 :   0.7644681   0.76517504
Instances de l'exemple 4 :   0.77163696   0.36197776
Instances de l'exemple 5 :   0.888528   0.13327008
```

Figure II.3. Affichage textuelle des données

3. Choisissez l'algorithme que vous vouliez utiliser

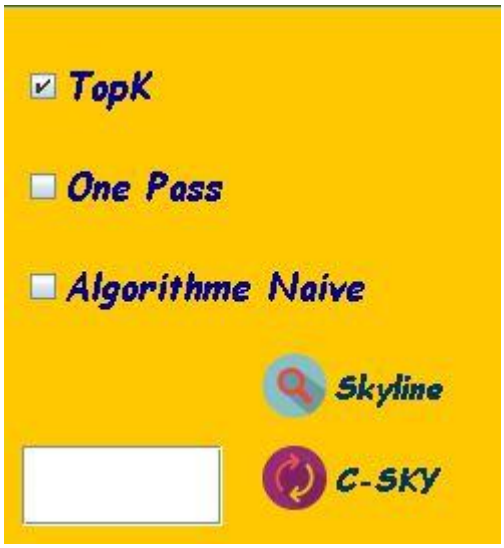


Figure II.4. Choix de l'algorithme à exécuter

3.1. Algorithme à base des requêtes top K :

Figure II.5. Configuration initiale pour l'algorithme à base des requêtes topK

```
Exemple de service 1
Instances de l'exemple 1 :  0.8364609  0.54697824
Instances de l'exemple 2 :  0.9935013  0.4124995
Instances de l'exemple 3 :  0.22692424  0.35843855
Instances de l'exemple 4 :  0.7607037  0.4061253
Instances de l'exemple 5 :  0.16229993  0.7874721

Exemple de service 2
Instances de l'exemple 1 :  0.21611995  0.73402125
Instances de l'exemple 2 :  0.48227537  0.6878936
Instances de l'exemple 3 :  0.82284904  0.7558313
Instances de l'exemple 4 :  0.12234074  0.7872392
Instances de l'exemple 5 :  0.63644564  0.9348112

Exemple de service 3
Instances de l'exemple 1 :  0.3107903  0.16722965
Instances de l'exemple 2 :  0.54566216  0.60663843
Instances de l'exemple 3 :  0.55029714  0.21119022
Instances de l'exemple 4 :  0.62098086  0.030439556
Instances de l'exemple 5 :  0.8294044  0.25299805
```

Figure II.6. Affichage textuelle des données

La première étape est de faire la somme de la QoS pour chaque classe et ensuite faire le tri de ces derniers.

```
Agrégation des critères :
TopClasse 1 [1] = 0.5689519   TopClasse 1 [2] = 1.2063975   TopClasse 1 [3] = 1.3851507
TopClasse 2 [1] = 0.55699563   TopClasse 2 [2] = 1.220216   TopClasse 2 [3] = 1.5938256
TopClasse 3 [1] = 1.179361   TopClasse 3 [2] = 1.098895   TopClasse 3 [3] = 1.0181413

Agrégation des critères après tri :
TopClasse 1 [1] = 1.3851507   TopClasse 1 [2] = 1.2179108   TopClasse 1 [3] = 1.2063975
TopClasse 2 [1] = 1.670505   TopClasse 2 [2] = 1.5938256   TopClasse 2 [3] = 1.220216
TopClasse 3 [1] = 1.3017493   TopClasse 3 [2] = 1.179361   TopClasse 3 [3] = 1.098895
```

Figure II.7. Affichage textuelle des données après le tri

Ensuite vous devez donner le nombre de résultat que vous voulez afficher pour le calcul des Top Classe composés et les trier.

```
Candidats C-TOP de chaque classe :
Candidat C-Top de la classe 1 :
TopClasse 1 [1] = 1.3851507   TopClasse 1 [2] = 1.2179108
Candidat C-Top de la classe 2 :
TopClasse 2 [1] = 1.670505   TopClasse 2 [2] = 1.5938256
Candidat C-Top de la classe 3 :
TopClasse 3 [1] = 1.3017493   TopClasse 3 [2] = 1.179361
```

Figure II.8. Affichage textuelle des TopK pour chaque classe

```
Calcul des candidats C-TOP :
C-Top [1] = 4.357405   C-Top [2] = 4.235017   C-Top [3] = 4.2807255   C-Top [4] = 4.158337   C-Top [5] = 4.190165
Tri des candidats C-TOP :
C-TOP [1] = 4.357405   C-TOP [2] = 4.2807255   C-TOP [3] = 4.235017   C-TOP [4] = 4.190165   C-TOP [5] = 4.158337
```

Figure II.9. Calcul des candidats TopK

```
Les CTop composés :
C-Top [1] = 4.357405   C-Top [2] = 4.2807255
```

Figure II.10. Affichage textuelle des C-TopK

3.2.Algorithme One Pass :

Figure II.11. Configuration initiale pour l'algorithme One Pass

```
Nombre de classes : 2
Nombre d'instances par classe est : 4
Nombre de critères est : 2

Exemple de service 1
Instances de l'exemple 1 :  0.9186014  0.42811435
Instances de l'exemple 2 :  0.09225124  0.13294727
Instances de l'exemple 3 :  0.59525627  0.9265159
Instances de l'exemple 4 :  0.57386005  0.8452514

Exemple de service 2
Instances de l'exemple 1 :  0.23867977  0.18300182
Instances de l'exemple 2 :  0.05428064  0.6096823
Instances de l'exemple 3 :  0.78374314  0.8424136
Instances de l'exemple 4 :  0.352588  0.6357724
```

Figure II.12. Affichage textuelle des données

La première étape consiste à calculer les Skylines de chaque service.

```
Sky_Ann 1
Sky_Ann1[ 1] =0.9186014   Sky_Ann1[ 1] =0.42811435
Sky_Ann1[ 2] =0.59525627  Sky_Ann1[ 2] =0.9265159
Sky_Ann1[ 3] =0.57386005  Sky_Ann1[ 3] =0.8452514

Sky_Ann 2
Sky_Ann2[ 1] =0.78374314  Sky_Ann2[ 1] =0.8424136
```

Figure II.13. Calcul des Skylines pour chaque classe

Ensuite calculer les Skylines composés et à la fin nous utilisons la dominance pour éliminer les Skylines composés dominés.

```
CSky_AOP
CSky  = 1.7023445  CSky  = 1.270528
CSky  = 1.3789995  CSky  = 1.7689295
CSky  = 1.3576032  CSky  = 1.687665
```

Figure II.14. Calcul des compositions C-SkY

```
CSky_Ann[ 1] =1.7023445  CSky_Ann[ 1] =1.270528
CSky_Ann[ 2] =1.3789995  CSky_Ann[ 2] =1.7689295
```

Figure II.15. Affichage textuelle des C-SKY retournés

3.3.Algorithme naïve :

Figure II.16. Configuration initiale pour l'algorithme Naïve

```
Nombre de classes : 2
Nombre d'instances par classe est : 5
Nombre de critères est : 2

Exemple de service 1
Instances de l'exemple 1 :  0.9430183  0.81974643
Instances de l'exemple 2 :  0.8275596  0.0013397932
Instances de l'exemple 3 :  0.96798277  0.8351697
Instances de l'exemple 4 :  0.0086868405  0.5853216
Instances de l'exemple 5 :  0.90723974  0.21908253

Exemple de service 2
Instances de l'exemple 1 :  0.68449146  0.5771416
Instances de l'exemple 2 :  0.31252778  0.67188025
Instances de l'exemple 3 :  0.41911572  0.27378637
Instances de l'exemple 4 :  0.4789412  0.6628505
Instances de l'exemple 5 :  0.98800415  0.98099
```

Figure II.17. Affichage textuelle des données

La première étape est de calculer les différents Skylines composés.

```
Liste des candidats skylines composées
CSky-AN [1][1] =1.6275098   CSky-AN [1][2] =1.396888
CSky-AN [2][1] =1.2555461   CSky-AN [2][2] =1.4916267
CSky-AN [3][1] =1.362134   CSky-AN [3][2] =1.0935328
CSky-AN [4][1] =1.4219595   CSky-AN [4][2] =1.4825969
CSky-AN [5][1] =1.9310224   CSky-AN [5][2] =1.8007364
CSky-AN [6][1] =1.5120511   CSky-AN [6][2] =0.5784814
CSky-AN [7][1] =1.1400874   CSky-AN [7][2] =0.67322004
CSky-AN [8][1] =1.2466753   CSky-AN [8][2] =0.27512616
CSky-AN [9][1] =1.3065008   CSky-AN [9][2] =0.6641903
CSky-AN [10][1] =1.8155637   CSky-AN [10][2] =0.9823298
CSky-AN [11][1] =1.6524742   CSky-AN [11][2] =1.4123113
CSky-AN [12][1] =1.2805105   CSky-AN [12][2] =1.5070499
CSky-AN [13][1] =1.3870986   CSky-AN [13][2] =1.1089561
CSky-AN [14][1] =1.446924   CSky-AN [14][2] =1.4980202
CSky-AN [15][1] =1.955987   CSky-AN [15][2] =1.8161597
CSky-AN [16][1] =0.6931783   CSky-AN [16][2] =1.1624632
CSky-AN [17][1] =0.32121462   CSky-AN [17][2] =1.2572019
CSky-AN [18][1] =0.42780256   CSky-AN [18][2] =0.859108
CSky-AN [19][1] =0.48762804   CSky-AN [19][2] =1.248172
CSky-AN [20][1] =0.996691   CSky-AN [20][2] =1.5663116
CSky-AN [21][1] =1.5917312   CSky-AN [21][2] =0.7962241
CSky-AN [22][1] =1.2197676   CSky-AN [22][2] =0.8909628
CSky-AN [23][1] =1.3263555   CSky-AN [23][2] =0.4928689
CSky-AN [24][1] =1.3861809   CSky-AN [24][2] =0.88193303
CSky-AN [25][1] =1.8952439   CSky-AN [25][2] =1.2000725
```

Figure II.18. Calcul des compositions C-SkY

Ensuite nous utilisons la dominance pour éliminer les Skylines composés dominés.

```
Liste des skylines composées
CSky_Ann [1][1]   =1.955987  CSky_Ann [1][2]   =1.8161597
```

Figure II.19. Affichage textuelle des C-SKY retournés

V. Expérimentation

Nous avons mené une expérience pour évaluer la performance de notre application. Le but de cette application est de démontrer comment on peut obtenir des points meilleurs. Nous avons développé notre prototype sous NetBeans IDE de Sun Microsystems, la machine d'expérimentation possède les caractéristiques suivantes :

- La machine DELL INSPIRON N5110.
- Le système d'exploitation est Windows 8 (64 Bits).
- Processeur Intel® Core™i7-2670QM CPU @2.20 GHz 2.20 GHz.
- 8 GB de RAM.

Nombres de classes	Nombres de critères	Nombres d'instances	Temps d'exécution (MS)
2	2	10	2
2	3	50	4
3	2	10	8
3	3	50	10
4	2	10	15
4	3	50	53

Tableau II.9. Expérimentation de l'algorithme TopK

Nombres de classes	Nombres de critères	Nombres d'instances	Temps d'exécution (MS)
2	2	10	16
2	3	50	1030
3	2	10	25
3	3	50	51745
4	2	10	90
4	3	50	89054

Tableau II.10. Expérimentation de l'algorithme One Pass

Nombres de classes	Nombres de critères	Nombres d'instances	Temps d'exécution (MS)
2	2	10	36
2	3	50	55987
3	2	10	223
3	3	50	61443
4	2	10	2158
4	3	50	135225

Tableau II.11. Expérimentation de l'algorithme naïve

Nous remarquons que le temps d'exécution dépend des paramètres initiaux pour les 03 algorithmes, si le nombre de classe est assez petit nous remarquons que le temps d'exécution est presque négligeable, nous soulignons aussi que le nombres d'instances

joue un grand rôle pour le temps d'exécution comme par exemple la différence entre le 5ème est le 6ème exemple est grande environ 135000 ms alors que nous avons le même nombre de classe.

Nous remarquons aussi que l'algorithme naïve est le moins performant par rapport au deux autres algorithmes, le Top k est performant mais le problème qui se pose avec cet algorithme est que certes les valeurs retournées sont bonnes mais ne sont pas considérer comme des Skylines et l'algorithme One Pass a une meilleure performance par rapport au deux autres algorithmes que ça soit avec des petites valeurs ou des grandes valeurs initiales.

VI. Conclusion

Dans ce chapitre nous avons présenté 03 algorithmes de recherche des skylines composés qui se base sur la QoS, ensuite nous avons fait une comparaison entre les performances de ses algorithmes.

Conclusion générale

Conclusion générale

Dans ce travail nous avons étudié la problématique de sélection de services web composés à base de la QoS, nous avons aussi montré les différentes approches de la sélection de services web à base de Qos, Nous avons aussi parler des différents domaines d'application des services web.

Dans la contribution nous avons proposé 03 algorithmes : l'algorithme basé sur les TopK services, l'algorithme à une seule passe (basé le calcul des skylines de chaque classe) et l'algorithme de recherche exhaustive (aussi appelé naïve). Nous avons aussi montré des expérimentations pour chaque algorithme et des comparaisons des 03 approches proposées afin de les évaluer.

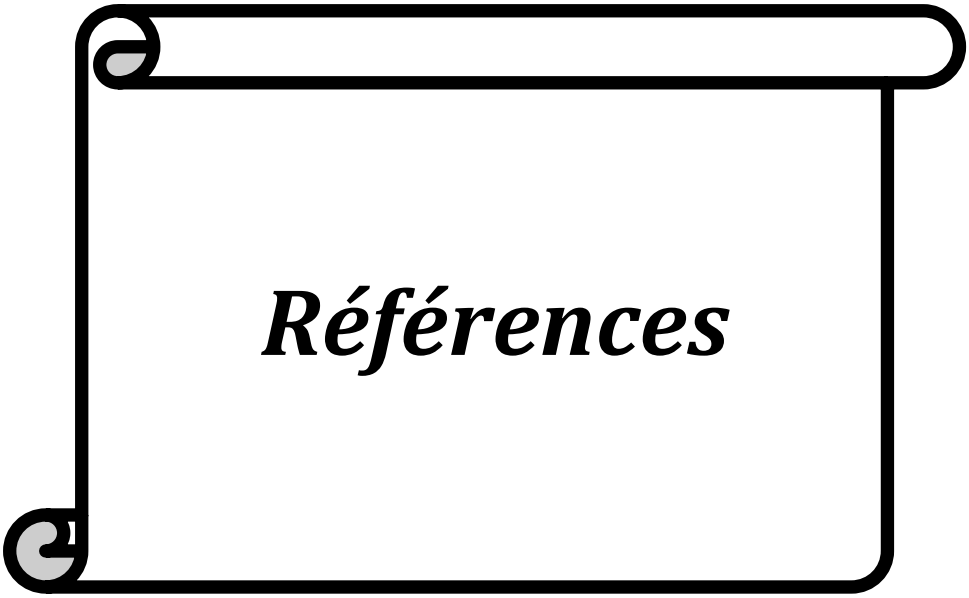
Dans nos futurs travaux nous envisageons d'ajouter des heuristiques du classement pour l'algorithme à une seule passe à fin d'avoir des résultats meilleurs, nous voulions aussi développer d'autres algorithmes performants qui se basent la qos , tels que « buttom up algorithm » et les comparer avec les approches que nous avons proposé.

Liste des figures

Liste des figures

Liste des tableaux

Liste des tableaux


Références

Références

[1] M. Alrifai, T. Risse, and W. Nejdl, A Hybrid Approach for Efficient Web Service Composition with End-to-End QoS Constraints. ACM Transactions on the Web, Vol. 6, No. 2, Article 7, 2012.

[2] A.Bouguettaya, Qi Yu, Efficient Service Skyline Computation for Composite Service Selection.

[3] B.Tassadit, analyse multidimensionnelle interactive de résultat de simulation.Aidea la décision dans le domaine de l'argoécologie, Thèse /Université deRennes 1 ,28 novembre 2013.

[4] F.Hadjila, M.A Chikh. QoS-aware Service Selection Based on Mimetic Algorithm. In Proceedings of CNEDI '12 Skikda. Algerie. 2013.

[5] F.Hadjila, M.A Chikh. QoS-aware Service Selection Based on Tabu Serach. In Proceedings of JEESI '12. Alger. Algerie. 2012.

[6] Q. Yu and A. Bouguettaya. Computing service skylines over sets of services. In ICWS, pages 481–488, 2010.

[7] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web, 1(1):6, 2007.

[8] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality-driven Web Service Composition. In WWW, 2003.

[9] S. Zhang, N. Mamoulis, and D. W. Cheung. Scalable skyline computation using object-based space partitioning. In SIGMOD, 2009.

[10] L. Zhu, Y. Tao, and S. Zhou. Distributed skyline retrieval with low bandwidth consumption. IEEE Trans. on Knowl. and Data Eng., 21:384–400, March 2009.

Résumé

Le but de ce projet est de proposer une solution pour accélérer la recherche des compositions de service web à base de la QoS. Pour ce faire nous proposons dans ce travail 03 approches articulées sur le concept de la QoS : la première approche est basée sur les requetes ToPK (ie. les fonctions d'agrégation des QoS, la deuxième est une approche naïve qui calcul toutes les compositions possibles ensuite elle élimine les éléments dominés, la troisième solution nommée « One Pass » calcule tout d'abord les Skylines de chaque classe ensuite elle extrait les compositions optimales à partir du résultat de la première phase.

Mots Clés : *méthodes de sélection, service web, composition de service web, dominance, Topk, Algorithme naïve, One Pass, Skyline, Skyline Composé, QoS.*

Abstract

The aim of this project is to accelerate the search of QoS aware web service compositions. To do this end, we propose in this work 03 approaches: the first one is based on TopK queries, the second one is an exhaustive search, and third one (termed one pass) leverages both local skylines and exhaustive search to fulfill the same purpose.

Keywords: *Selection methods, Web service, Composite web service, TopK, Naïve algorithm, One Pass algorithm, Skyline, Composite Skyline, QoS.*

ملخص

الهدف من هذا العمل هو اسراع البحث عن تشكيلات خدمات الويب مبنية على جودة الخدمة.من أجل ذلك إقترحنا 03 طرق كلها مبنية على أساس جودة الخدمة، الطريقة الأولى نقوم بعملية جمع وذلك لجميع الأمثلة وبعد ذلك نقوم بترتيبها وحساب جميع التشكيلات ثم نقوم بترتيبها هي لأخرى وفي أخير نعرض النتائج حسب طلب المستخدم، أما الطريقة الثانية فهي طريقة سذجة حيث تقوم بحساب جميع التشكيلات الممكنة وبعد ذلك تقوم بنزع جميع التشكيلات التي يوجد على الأقل تشكيلة واحدة أفضل منها، أما الطريقة الثالثة فهي في هذه الطريقة نقوم باستخراج جميع الخدمات التي لا يوجد أحسن منها وذلك لجميع القسام ثم نقوم بحساب جميع التشكيلات الممكنة ثم ننزع التشكيلات التي يوجد على الأقل ثشكيلة أفضل منها

الكلمات الرئيسية: *:طرق الإختيار، جودة الخدمة، خدمات الويب، تشكيلات خدمات الويب*