Lo scopo primario dell’algoritmo progettato è quello di suddividere per tipologia un insieme finito di pagine dato in input. Tale suddivisione per tipologia è necessaria al fine di consentire ad un estrattore di dati di estrarre per ogni tipologia solamente il contenuto rilevante.

L’algoritmo è suddivisibile in 3 principali parti.

**PARTE I**

Il primo passo che viene eseguito è un estrazione di regole XPath su ognuna delle pagine presenti in input. Tale estrazione è stata delegata ad un algoritmo fornitomi e sviluppato dal Professor Crescenzi. Vengono poi eliminate le regole che risultano essere equivalenti, sia in termini di regola stessa, che di valori estratti sulla pagina dalla regola.

*Set<XPath> rulesGeneration(Set<Page> pages){*

*Map<Page, Set<String>> p2x*

*Set<String> diffRules*

*for (page : pages){*

*Set<String> rules = inferRules(page)*

*diffRules.addall(rules)*

*p2x.put(page,rules)*

*}*

*Set<XPath> xpaths;*

*for (rule : diffRules){*

*Set<Page> rulePages;*

*for(page : p2x.key()){*

*if(p2x.get(page).contain(rule))*

*rulePages.add(page)*

*}*

*xpath = new XPath(rule,rulePages);*

*xpaths.add(xpath)*

*for(page : rulePages)*

*page.addXPath(xpath)*

*}*

*return xpaths;*

*}*

Si procede quindi al raggruppamento delle pagine in vari gruppi, chiamati classi di pagine. Tali classi di pagine raggruppano pagine che condividono regole XPath che sono comuni solo a quel preciso gruppo e a nessun altro. Ognuna di tali regole, specifica per una singola classe di pagine, è detta XPath caratteristico.

*Set<PageClass> groupPagesByXPaths(Set<XPath> xpaths){*

*Set<PageClass> pageClasses*

*for(xpath : xpaths){*

*Set<XPath> pageClassRules*

*pageClassRules.add(xpath)*

*for(xpath2 : xpaths){*

*if (xpath != xpath2 && checkSamePages(xpath,xpath2)){*

*pageClassRules.add(xpath2)*

*}*

*}*

*pageClass = new PageClass(xpath.getPages(), pageClassRules)*

*if(!pageClasses.containsXPathsSet(pageClass)*

*pageClasses.add(pageclass)*

*}*

*return pageClasses*

*}*

Vengono poi formate delle partizioni a partire dalle classi di pagine ottenute. Ogni partizione è un insieme di classi di pagine tale che ogni pagina contenuta nell’insieme iniziale dato in input sia presente in una e una sola delle classi di pagine selezionate. Vengono formate tutte le partizioni che risultano realizzabili date le classi di pagine ottenute. Tali partizioni vengono poi organizzate in un reticolo detto lattice sulla base di un ordinamento parziale, detto raffinamento.

*Lattice generatePartitions(Set<PageClass> pageClasses, int MAX\_PAGES){*

*Set<Partition> partitions*

*Set<PageClasses> toCheck = pageClasses*

*for( pageClass : pageClasses){*

*toCheck.remove(pageClass)*

*Set<PageClass> current*

*current.add(pageClass)*

*partitions.addall(allPossiblePartitions(current,toCheck,*

*pageClass.getPages().size(),MAX\_PAGES)*

*}*

*return new Lattice(partitions)*

*}*

*Set<Partition> allPossiblePartitions(Set<PageClass> currentClasses,Set<PageClass> toCheck, int pageCount,int max){*

*Set<Partition> partitions*

*if(pageCount == max){*

*partition = new Partition(currentClasses)*

*partitions.add(partition)*

*}*

*else if(!toCheck.isEmpty){*

*Set<PageClass> newToCheck*

*newToCheck.addAll(toCheck)*

*for(pageClass : toCheck){*

*newToCheck.remove(pageClass)*

*if( (pageCount + pageClass.getPages().size() <= max) && + (!PageClass.hasSamePagesAs(currentClasses))){*

*Set<PageClass> newCurrent*

*newCurrent.addAll(currentClasses)*

*newCurrent.add(pageClass)*

*partitions.addAll(allPossiblePartitions(newCurrent,newToCheck,*

*(pageCount+ pageClass.getPages().size()),max))*

*}*

*}*

*}*

*return partitions*

*}*

**PARTE II**

A partire dallo struttura del reticolo generato si procede quindi alla ricerca di eventuali punti fissi navigazionali, detti NFP(Navigational fixed point). La ricerca viene effettuata per ognuna delle classi di pagine e vengono isolati i punti fissi, dopo una rimozione di eventuali collegamenti a pagine non facenti parte dell’insieme analizzato, che permettono la navigazione verso una delle altri classi di pagine. Identificato un collegamento verso una classe di pagine lo si ritiene valido anche per le classi di pagine di cui la classe trovata è raffinamento, poiché è possibile che da una determinata classe di pagine “ottima” non siano accessibili tutte le pagine facenti parte di un’altra classe di pagine “ottima” portando quindi a risultati con suddivisioni ulteriori delle classi di pagine desiderate.

*public void executeNFP(Set<PageClass> pageClasses, Set<Page> pages){*

*for(PageClass pageClass : pageClasses){*

*Map<Set<String>, int[]> NFP = NFP(pageClass,pages)*

*if (NFP!= null) {*

*for(Set<String> IDs : NFP.keySet()){*

*Set<PageClass> NavClasses = PageClass.getPageClassesFromIDs(pageClasses,pcID)*

*if(NavClasses != null)*

*if(!NavClasses.isEmpty()){*

*NavClasses.remove(pageClass)*

*pageClass.addNFP(NavClasses,NFP.get(IDs))*

*}*

*}*

*}*

*}*

*}*

Si procede quindi a valutare la presenza di eventuali punti fissi navigazionali nelle partizioni. Per ogni partizione vengono cercati i punti fissi che, generati su una classe di pagine appartenente alla partizione, conducono ad un’altra classe di pagine anch’essa appartenente alla partizione. I punti fissi navigazionali cosi trovati, detti punti fissi navigazionali interni ad una partizione, sono un efficace metodo per effettuare una prima selezione delle partizioni candidate ad essere “l’ottimo”. Si isolano dunque le partizioni che presentino il maggior numero di NFP, tenendo conto in differente misura di punti fissi costanti e variabili.

*for (Partition p : lattice.getPartitions()){*

*int[] total;*

*for(PageClass pageClass : p.getPageClasses){*

*for(Set<PageClass> pointsTo : pageClass.getNFP().keySet()){*

*for(PageClass pageClass2 : pointsTo){*

*if(p.getPageClasses().contains(pageClass2)){*

*total += pageClass.getNFP().get(pointsTo)*

*}*

*}*

*}*

*}*

*}*

**PARTE III**

Si procede quindi alla ricerca di punti fissi di dati, DFP(Data fixed point) sulle rimanenti partizioni. Vengono ricercati i punti fissi per ogni classe di pagine che compone le partizioni con il maggior numero di NFP. Vengono selezionati solamente i punti fissi che generano valori per ogni pagina della classe analizzata, si trascurano quindi i punti fissi parziali, comuni solo ad alcune delle pagine, detti opzionali. Si rimuovono inoltre dai punti fissi trovati quelli comuni ad ognuna delle pagine prese inizialmente in input, detti punti fissi di sito.

*public void executeDFP(Set<PageClass> pageClasses, Set<Page> pages, Set<FixedPoint<String>> siteDFP){*

*Map<FixedPoint<String>, PageClass> FP2PC*

*for(PageClass pc : pageClasses) {*

*Set<FixedPoint<String>> FixedPoints = DFP(XFParguments,pc,pages,range);*

*for(FixedPoint<String> fp : FixedPoints) {*

*if(!siteDFP.contains(fp)) {*

*FP2PC = addFixedPoints(FP2PC, fp, pc);*

*}*

*}*

*}*

*addToPageClass(FP2PC);*

*}*

*}*

Si assegna dunque ad ognuna delle partizioni un punteggio dato dalla media di punti fissi trovati nelle classi di pagine che la compongono, tenendo sempre conto di una differenza di rilevanza tra punti fissi costanti e variabili ed escludendo nuovamente eventuali punti fissi “opzionali”. L’ottimo corrisponde alla partizione con il punteggio più alto.