Recommending Improvements to Web Applications Using Quality-Driven Heuristic Search

Stephane Vaucher[1], Samuel Boclinville[2], Houari Sahraoui[1], and Naji Habra[2]

[1] DIRO, Université de Montréal, Québec, Canada
{vauchers, sahraouh}@iro.umontreal.ca
[2] PReCISE Research Center, FUNDP, University of Namur, Belgium
samuel.boclinville@student.fundp.ac.be, nhag@info.fundp.ac.be

Abstract. Planning out maintenance tasks to increase the quality of Web applications can be difficult for a manager. First, it is hard to evaluate the precise effect of a task on quality. Second, quality improvement will generally be the result of applying a combination of available tasks; identifying the best combination can be complicated. We present a general approach to recommend improvements to Web applications. The approach uses a meta-heuristic algorithm to find the best sequence of changes given a quality model responsible to evaluate the fitness of candidate sequences. This approach was tested using a navigability model on 15 different Web pages. The meta-heuristic recommended the best possible sequence for every tested configuration, while being much more efficient than an exhaustive search with respect to execution time.

1 Introduction

Successful Web applications (WAs) not only satisfy the need of their users for interesting features, but also provide a pleasant user experience as well. Quality problems (eg. bad navigability) can have a negative impact on this experience, and should therefore be corrected. For a manager planning out the maintenance tasks, identifying which task can best correct such problems can be daunting since its impact on quality is generally unknown beforehand.

Many quality characteristics can be defined in a model that evaluates the extent to which an application conforms to a set of non-functional requirements. Such a model typically includes quality rules/standards and is refered to as a *quality model*. While the use of (implicit or explicit) quality models should be part of any improvement process, quality models do not allow, by themselves, to determine what specific changes should be implemented to improve the quality of an application. This is reflected in the literature. There are many contributions focussing on the quality *assessment* of Web applications, yet to the best of our knowledge there is little or no work focussing on its improvement, or more exactly, how to achieve a certain level of quality given resource limitations. What work exists is not specific to Web applications, but rather to general software development [1,2,3].

G. Vossen, D.D.E. Long, and J.X. Yu (Eds.): WISE 2009, LNCS 5802, pp. 321–334, 2009.

In this article, we present a method to propose improvements to a Web application on the basis of a quality model. Given a model, a set of possible transformations and an estimate of available resources, our method will propose an optimised sequence of transformations to apply to an application. To find the exact sequence is generally intractable since it is not solvable in polynomial time. We therefore propose the use of meta-heuristics to find a suitable approximation. The method is implemented using the *simulated annealing* meta-heuristic and is tested in a study that serves as a proof of concept. The study evaluates improvements proposed to the navigability of 15 Web pages. The proposed method is shown to identify the optimal solution in constant time relative to the number of transformations considered. The method is therefore deemed to be well-suited for this problem.

The paper is organised as follows. In the next Section, we present related work in both Web quality assessment and quality improvement by refactoring. Section 3 presents an overview of the proposed method. The quality model used for the study is presented in Sect. 4. Section 5 details the meta-heuristic algorithm (simulated annealing) and the transformations used. Finally, Sect. 6 presents the case study.

2 Related Work

Our work crosscuts two research domains Web quality and refactoring. We present work related to our problem in both domains.

To evaluate WA quality, several contributions have proposed guidelines, principles, checklists, evaluation methodologies and automatic assessment tools. We present some of these contributions. Nielsen and Loranger [4] propose advice on how to address Web usability issues. Boldyreff et al. [5] present a set of metrics to assess WA quality and particularly WA evolvability. Deluze [6] discusses WA performance with respect to existing Internet technologies. These contributions give principles but do not explicitly show how to use them to evaluate the quality of a WA.

Some authors proposed quality assessment methods or models such as WebQEM (Web Quality Evaluation Methodology) by Olsina et al. [7] or FMSQE (Fuzzy Model for Software Quality Evaluation) by Albuquerque et al. [8] . Like many others, these contributions define quality in tree-like, hierarchical models inspired by the ISO 9126 software product standard. The evaluation of a high-level notion of quality is done by aggregating the evaluations of more concrete (and more measurable) sub-factors.

Directly related to this work, Malek et al. [9] proposed a method for building WA quality models using Bayesian networks. Following this method, they produced a model to measure the navigability of Web pages, presented in [15]. In an controlled experiment, they showed that the model could accurately measure the notion of navigability as perceived as by a user. We therefore reused this model to go beyond simple quality assessment and suggest actual improvements to pages. The model is briefly described in Sect. 4. The approach of Malek et al. was also used by Caro et al. [10] for the particular case of Web portal data quality.

Some work has been done to bridge the gap between quality assessment and improvement, mostly in the field of object-oriented (OO) software development. In particular, the focus was put on the impact of refactoring on OO metrics (as in Sahraoui et al. [1] and Dubois et al. [11]). The effect on quality is however implicit (e.g. lower coupling is good) and does not take into consideration a precise and complete evaluation according to a quality model. The determination of refactoring sequences using heuristic-search methods was studied for OO programs (e.g. Seng et al. [3], and Harman and Tratt [12]). In general, the objective is to determine the transformation sequences that best improve some design metrics. This work is however done without knowing whether or not this metric selected is indeed a measure of quality.

Finally, for Web refactoring, Olsina et al. [2] propose refactoring patterns that can be applied to design models with the perspective of improving quality. Ping and Kontogiannis [13] propose a refactoring approach that addresses specifically WA architecture, *i.e.,* dependencies between Web pages. For both contributions, the link between refactoring and quality is implicit, and yet again, the notion of quality is based on best practices instead of on a well-defined notion of quality as provided by a quality model.

3 Recommending Improvements

Any recommendation should be the result of a cost-benefit analysis which compares the *cost* of implementing a change with the *benefit* measured in terms of quality improvement. Although managers should be accustomed to estimating the cost of a change, it is less obvious to estimate its effect on a quality. In fact, most modifications (e.g. refactoring) have a theoretical impact on one or several quality criteria, but it is difficult to decide which of a set of possible modifications is the most appropriate given a specific context. Furthermore, this decision process is even more complicated when these modifications are combined. In this Section, we present how our approach solves these problems.

3.1 Global Overview

At the heart of the proposed approach is the concept of transformations. A transformation is a simple modification to a WA that corresponds to a well-defined developer activity. This activity has a cost and affects the state of a Web page. The objective is to select, out of a domain of possible transformations, the best sequence to apply to a given Web page. In the proposed approach, the cost is determined using an experts estimation of work while the impact of a given solution is left to a quality model.

The proposed approach is illustrated in Fig. 1. It includes two processes used for recommendation: first, the evaluation of quality and second, the suggestion of transformation. The evaluation process judges a page in a given state: the current state, or the state after a sequence of transformations. This process is used by the suggestion process to find an optimal solution.

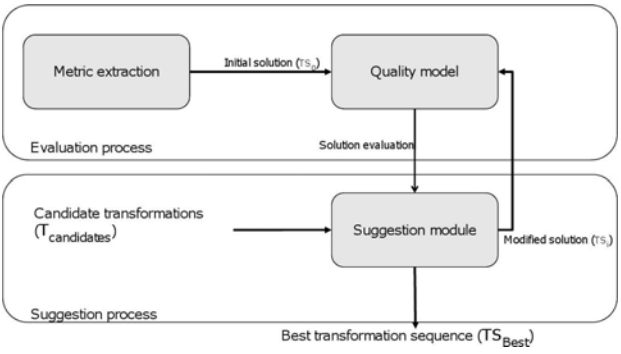

Fig. 1. The global process for the improvement of Web application quality

3.2 Quality Evaluation Process

To evaluate the quality of a page, we rely on a quality model (QM). There are many advantages to relying on a QM. First, it automates the judgment of an expert and can decide objectively even in the presence of contradictory information (ex: clutter vs functionality). Second, as presented in Sect. 2, there are many existing models documented present in the literature. It is thus possible to reuse previous work.

If we take the example of a transformation that adds a search engine to a Web site. The model would not only consider a search functionality has been added, but would likely also consider the effect of the new links/forms necessary to access the the search page. The effect of applying this transformation on the quality of a Web page is not obvious. If we consider a QM that models navigability, adding a search engine might improve navigability in a Web site, but can also add clutter to an interface, decreasing the navigability of a page. In these cases, QMs automates the decision of whether or not the quality has been globally improved.

A model uses metrics as inputs describing the state of a Web page and outputs a score. If we consider a vector of metric values $m = (m_1, m_2, ...)$ used as inputs to a model, the quality model is basically a function $q(m)$.

3.3 Suggestion Process

This process is responsible for suggesting the best sequence of transformations (called solution) that can be applied to a Web page. The fitness of a solution is judged by its effect on its quality as measured by the quality model. Since the model uses metrics as inputs, transformations are defined according to their effects on these metric. Finally, there is a cost constraint limiting the transformations that can be considered.

Measuring Quality Improvement. Let TS be a transformations sequence $TS = (t_1, t_2, ..., t_n)$ *i.e.*, an ordered sequence composed of elements taken from

a set of available transformations ($T_{candidates}$). The impact of such a transformation sequence can be expressed as a function that modifies the vector of metric values m by cumulating the effect of the transformations $t_1, t_2, ..., t_n$ in that order.

This function, $apply\_transform$ (1), computes new inputs metrics given a transformation sequence TS. Each individual transformation t_i in TS, is associated to a metric transformation function tr_met_i corresponding to the effect of applying that transformation on the Web page. The output of $apply\_transform$ is the successive application of the functions of the transformations it contains. The modified metrics (m') are then used by the QM to evaluate the global score of the modified page ($q(m')$).

$$m' = apply\_transform(TS, m) = tr\_met_n(tr\_met_{n-1}(\cdots tr\_met_1(m)) \quad (1)$$

Selecting Solutions. The suggestion process should identify the best solution (TS_{best}) which maximises the quality improvement (Equation 2) given a cost constraint (W).

$$\forall(TS_i) : q(apply\_transform(TS_{best}, m_0)) > q(apply\_transform(TS_i, m_0)) \quad (2)$$

Each transformation t_i of TS has a cost associated to it, $cost(t_i)$ that should reflect the cost of implementing the transformations. The global cost of a transformation sequence is the sum of the costs of the different transformations composing the sequence and should be inferior to a cost constraint W.

$$Cost(TS) = \sum_{1}^{n} cost(t_i) < W \quad (3)$$

Searching for the Best Solution. To find an optimal solution, the suggestion process needs to consider every possible solution. Without the cost constraint, the search space is nearly infine because some transformations can be repeated. Even with the cost constraint, the search space cannot be explored in polynomial time. If we were to simplify the problem and consider that every possible transformation can be applied or not (binary decision), with no ordering, the number of solutions to consider is $\in O(2^n)$. Furthermore, since there is no direct relationship between a transformation and its effect on quality, we cannot limit the search space. There are however meta-heuristics that can find nearly optimal solutions by guiding the exploration of the large search space in a reasonnable amount of time [14].

4 Assessing Web Quality

In this Section, we present the kind of quality model used by the suggestion process to evaluate proposed transformation sequences. This general description is accompanied by the description of a model focussing on the evaluation of a major attribute, the navigability of Web pages.

4.1 Quality Model

Quality models are built to systematize a qualitative judgment by a stakeholder of a phenomenon. For these QM to be useful, they should be as automated as possible. Generally, they use metrics to quantitatively describe an observable state as perceived by an expert and output a judgment as either a predicted category (e.g. good vs. bad) or a predicted value (e.g. score).

For use in the proposed approach, a QM must return a global score which can be used to compare different solutions. Furthermore, its judgment process must be fully automated, requiring no information beyond the initial description of a Web page.

4.2 Navigability Model

To illustrate our recommendation approach, we use the QM presented in [15]; it evaluates the navigability of Web pages. The QM is built using Bayesian Belief Networks (BBNs). BBNs organise an evaluation process as a graph. The graph contains links which indicate causal or descriptive relationships and nodes which indicate metrics or decisions. The metrics are the inputs to the model and are represented using probability distributions. Nodes with incoming links are decisions (intermediate or final) and are represented using conditional probability tables.

Figure 2 presents the evaluation procedure for navigability. The model has 13 inputs which are described in Table 1. The decision of whether or not a page is navigable (navigability node) directly depends on three sub-characteristics: the ability of a user to *locate* a desired page, his ability to find information on the desired page (bind) and the download speed. Both the locate and bind nodes are intermediate decision nodes which depends on other sub-characteristics.

The operational details of the QM including the precise process to convert metrics into probability distributions is described in [15]. In a nutshell, binary

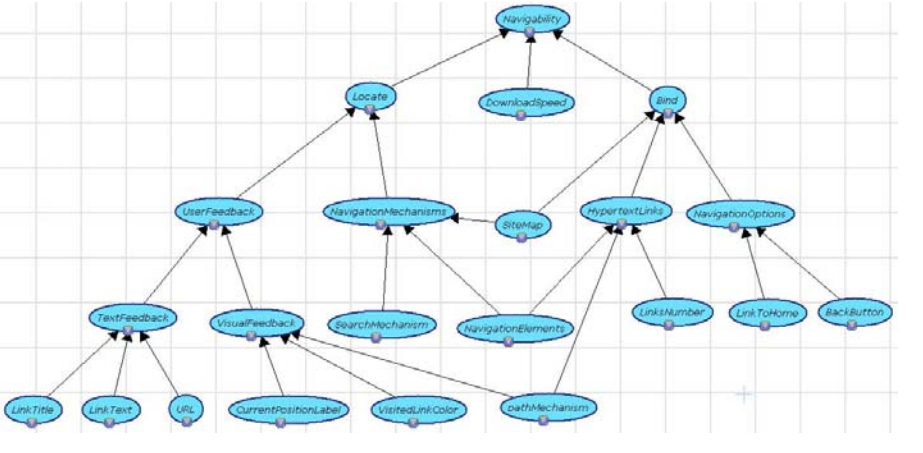

Fig. 2. The navigability model

metrics are converted into probabilities of 0 or 1 if the metric is respectively false or true. Numeric metrics are converted into discrete probabilities distributions (e.g. for $P(low)$, $P(medium)$ or $P(high)$) in two steps. First, fuzzy clustering is used to identify significant thresholds for the metrics values (a threshold for every possible discrete value). Second, when evaluating a Web page, the probability distribution for input nodes is interpolated using the distance of the metrics extracted with the neighbouring thresholds.

The output node (Navigability) corresponds to the probability that a user would consider the navigability satisfactory. The more likely users are satisfied, the better the quality of the page.

Table 1. Inputs to the navigability model

Metric	Node	Type
Download speed	DownloadSpeed	measure (count s.)
Ratio of links with titles (LTitle)	LinkTitle	measure ([0,1])
Ratio of links with text (LTxt)	LinkText	measure ([0,1])
Presence of a search engine (SE)	SearchMechanism	binary
Presence of a site map (SM)	SiteMap	binary
Presence of a menu (Nav)	NavigationElements	binary
Indication of location in Web site (CPL)	CurrentPositionLabel	binary
Visited links change color (VLC)	VisitedLinkColor	binary
Breadcrumbs (BRC)	PathMechanism	binary
Number of links in page (NoL)	LinkNumber	measure (count)
Link to home (Home)	LinkToHome	binary
Support for Back Button (BB)	BackButton	binary

5 Suggesting Improvements

In this Section, we present the suggestion process; this process is illustrated using the example of recommending improvements to navigability. To begin, transformations are defined with respect to their effect on the inputs of the navigability QM presented earlier. Then, we present how to select the best sequence of transformations to improve the quality of a Web page. Considering every possible solution when there are many possible transformations is intractable. We therefore rely on a meta-heuristic algorithm: *simulated annealing*.

5.1 Transformations for Navigability

To improve quality, it is necessary to first identify the set of candidate transformations ($T_{candidates}$) and define their effects on the inputs of the QM. Let T_{NAV} be the set of candidate transformations available to improve navigability. These transformations can be general changes used to improve quality like refactorings or quality-specific changes (e.g. adding a search engine). In Table 2, we define a set of transformations applicable to improving navigability of a Web page. They

Table 2. Selected transformations (T_{NAV}) and their effects on metrics(Table 1))

Transformation	Metric variation
Add/Remove site map	NoL +/- 1, SM := true/false
Add/Remove search engine	NoL +/- 1, SE := true/false
Add/Remove URL*	NoL +/- 1
Add/Remove link to Home	NoL +/- 1, Home := true/false
Add/Remove menu	NoL +/- a, Nav := true/false
Add/Remove current position label	NoL +/- b, CPL := true/false
Add/Remove breadcrumbs	NoL +/- c*, BRC := true/false
Enable/Disable back button support	BB := true/false
Add/Remove visited colour link	VLC := true
Divide/Merge page*	NoL := NoL ÷/× 2 +/- d*
Correct link text	LTxt := 100%
Correct/Remove link titles	LTitle := 100%/0%

are defined according to their effect on the 12 metrics of the navigability QM. Those which can be repeated multiple times are identified by an asterisk.

Some transformations like adding/removing breadcrumbs affect a variable number of links (variables a, b, c and d) which depends on the Web applications themselves. The exact number of links need to be evaluated empirically.

5.2 Simulated Annealing

A naive approach to suggestion would consider every possible transformation sequence given a cost constraint. Although the small size of T_{NAV} allows for an exploration of every possible solution in reasonnable time, it would not scale if we were to consider a larger set of transformations.

Simulated annealing(SA) is a meta-heuristic algorithm inspired by the metallurgic process of annealing. It guides an exploration of a large search space to find a mear optimal solution. It follows the steps described in Algorithm 1. It reflects a cooling process where a temperature Trs_0 decreases geometrically by a factor α at every iteration. Every iteration, a solution is compared to Nrs neighbouring solutions. When a neighbouring solution is better, as judged by a fitness function, it is retained for the next iteration. If the neighbour is not better, it can still be retained if, but only with a probability that depends on the current temperature. This element of randomness exists to avoid stagnating at a locally optimal solution. As the algorithm temperature drops, there however are fewer weaker solutions accepted as $e^{-\Delta/T_{rs}}$ decreases. Finally, the algorithm ends when until a specified number of iterations has passed. As with all meta-heuristics, SA is a general algorithm which is adapted to specific problems. Three elements are required to adapt SA: the problem's search space, the neighbourhood function and the fitness function.

Solution Space and Representation. T_{NAV} defines the transformations available for any given page evaluated by the navigability QM. When evaluating a page, not all transformations are however applicable. Excluding these transformations limits the size of the search space and improves performance. For example, it is useless to consider correcting link titles for a page where the links are all correct. We therefore limit the domain of valid transformations for a specific Web page $T_{Page} \subset T_{NAV}$ and solutions are vectors of transformations $s = (t_1, t_2, ..., t_n)$ where $t_i \in T_{Page}$.

Neighbourhood Function. The initial state is an empty sequence of transformations and the neighbourhood function(NF) is responsible to iteratively modify it until a satisfactory final solution is found. For every iteration of the SA algorithm, the neighbourhood function chooses randomly to either add a new transformation to the solution or remove a transformation already contained in s. Possible additions are however limited to those that will preserve the coherence of the overall solution. The set of acceptable transformations is $T_{neigh} \subset T_{page}$ where $t_i \in T_{neigh} \wedge coherent(s, t_i)$ for a solution s. The coherence function serves not only to ensure that a solution returned makes sense to a user, but also to reduce the search space.

Two possible situations can lead to incoherent sequences: *redundant* transformations and *opposite* transformations. Some transformations affect discrete values of metrics of a quality model (e.g. adding a search engine). It would be redundant for these types of transformation to be reapplied twice. Some groups of transformations have opposite effects (e.g. merging and splitting a page). A candidate transformation is deemed incoherent with an existing solution if the solution already contains an opposite transformation. The only way it can be inserted is if all the opposite transformations are deleted first.

Fitness Function. For any given solution, a QM acts as a fitness function to evaluates how "good" is a solution. Since the SA algorithm requires a single value for the fitness evaluation of a solution, the fitness function needs to combine the notions of cost and quality improvement. Our fitness function (4) penalises a solution if it does not respect the cost constraint. A higher QM score solution which does not respect the cost constraint (W) will always yield a lower fitness value than any solution that respects it.

$$fitness(s) = \begin{cases} 0.5 + Nav\_Impr * 0.5 & \text{if cost(s)} \leq \text{W} \\ Nav\_Impr * 0.5 & \text{if cost(s)} > \text{W} \end{cases} \quad (4)$$

where Nav_Impr corresponds to the quality improvement: the relative variation between the current QM score and the initial score.

6 Case Study

In this Section, we present a study to evaluate the feasibility of applying the proposed approach in an industrial context.

```
SIMULATED ANNEALING(Trs0 ,Nrs, . . .) ;
Choose s_0                                   /* Chose initial solution */ ;
Trs := Trs_0                                 /* Initialise temperature */ ;
STOP := false ;
BestSol := s := s_0                          /* Initialise Best Solution */ ;
while !STOP do
    for i := 1 to Nrs do
        Generate s' ∈ N(s)                   /* Generate neighbour */ ;
        Δfitness := fitness(s') − fitness(s) /* Fitness function */ ;
        if ΔC ≥ 0 then
            s := s'                          /* accept the solution */ ;
        else
            Generate random number r ∈ [0, 1] ;
            if r ≤ e^(−ΔC/Trs) then
                s := s'                      /* Accept with small probability */ ;
            end
        end
        if C(s') ≤ C(BestSol) then
            BestSol := s' ;
        end
    end
    Trs := α.Trs                             /* Lower the temperature */ ;
    if StopCriteria then
        STOP := True ;
    end
end
return BestSol ;
```

Algorithm 1. Simulated Annealing

6.1 Objectives

The goal of the study is to verify the usefulness of the proposed recommendation approach from the perspective of manager facing quality problems. The approach is evaluated according to two objectives:

- Verify if the meta-heuristic finds the optimal solution;
- Measure if the execution uses a reasonable amount of time and resources. This is particularly important since the evaluation of a Web application would depend on the evaluation of every Web page.

We used the QM and the transformation set respectively presented in Sects. 4 and 5. For the first objective, we compared the quality improvement of the solution returned by an exhaustive search to that of the best solution of the SA algorithm. For the second objective, we compared their execution times. It was possible to compare our approach to an exhaustive search types because of the small number of possible transformations (T_{NAV}) considered in this case.

Table 3. Transformation domain and initial quality of studied pages

# Page	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\|T_{Page}\|$	21	17	17	15	13	13	15	15	9	11	13	11	9	11	9
q_{NAV}	13%	19%	24%	25%	68%	69%	73%	73%	76%	77%	83%	84%	84%	84%	85%

6.2 Study Setup

The comparison is performed on a set of 15 randomly selected Web pages. These pages are of different quality (q_{NAV}) and have different search spaces ($|T_{Page}|$) as shown in Table 3. As expected, the pages with fewer possible transformations generally have better navigability scores.

In order to collect metrics on the Web pages, we developed a Web application. Most metrics could be extracted automatically by using htmlunit[1], a Java library generally used to test Web sites. The metrics which required human intervention were entered using a Web interface. This interface also allows a user to filter out transformations that should not be considered and define T_{Page}.

The SA algorithm was initialised with the following parameters: $Trs = 10$, $Nrs = 30$ and $\alpha = 0.8$. The Trs value can be considered a typical value considering the size of the search space. At every iteration. the algorithm selects 30 random neighbours. The α value chosen is relatively low, as normal values are closer to 0.9. This parameter limits the number of iterations used.

Table 4. Transformation costs

Transformation	Cost
Add/Remove site map	3
Add/Remove search engine	6
Add/Remove URL	2
Add/Remove link to Home	1
Add/Remove menu	6
Add/Remove current position label	5
Add/Remove breadcrumbs	6
Enable/Disable back button support	1
Add/Remove visited colour link	2
Divide/Merge page	8
Correct link texts	5
Correct/Remove all the link titles	6/4

Setting Costs. To compare the quality solutions found (first objective), Web pages are evaluated according to four different cost constraints (W): $< 0.25W_{MAX}$, $< 0.5W_{MAX}$, $< 0.75W_{MAX}$ and $< W_{MAX}$, where W_{MAX} is the maximum cost of

[1] `http://htmlunit.sourceforge.net`

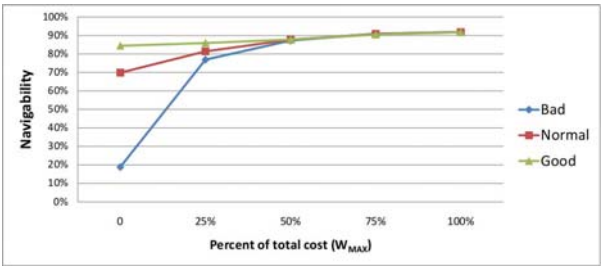

Fig. 3. Quality for different cost constaints

a solution. This cost is computed by adding the cost of all transformations $\in T_{Page}$. The costs of the different transformations, are set according to our expected level of difficulty. The costs are presented in Table 4. Whether these costs are realistic or not, should not affect the validity of the results since they affect both compared search types (SA and exhaustive) equivalently.

6.3 Results

For every Web page considered and for every cost constraint, the SA algorithm found the best solution. This verifies the first objective of the study. Fig. 3 summarises the quality improvement for each cost step. In this plot, pages are grouped together according to their initial quality scores to minimise visual clutter. There are pages with bad navigability ($q_{NAV} \leq 25\%$, pages 1-4), normal navigability (pages 5-10), and good navigability ($> 80\%$, pages 11-15).

After the first cost constraint (25% of total cost), quality is improved for all pages, most significantly for bad and normal pages (increase of 70% and 10% respectively) The reason for this is simple, the most cost-effective transformations are identified and executed first. The worst pages tend to improve much more than good pages at this step because they are more ways they can be transformed positively than good pages. Another explanation is that their sets T_{Page} tend to be larger and thus W_{MAX} (and consequently $0.25W_{MAX}$) is much

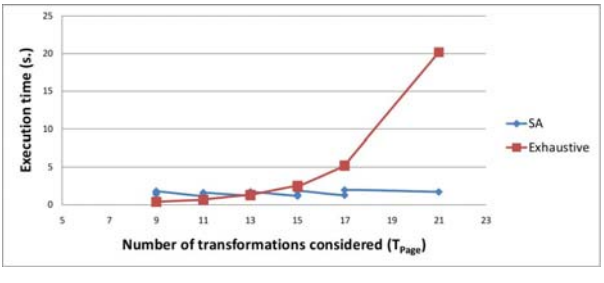

Fig. 4. Execution time vs. number of transformations considered

larger than that of better pages. From the second constraint on (50% of total cost), the quality of all pages becomes less and less discriminable as it gradually converges to the best possible quality level (92%), attained when ($W = W_{MAX}$).

Figure 4 shows the difference in execution time. The number of considered candidate transformations is on the X axis, while the execution time is on the Y axis. The Figure shows that SA requires constant (1.5 s.) time to find the best solution while the exhaustive search quickly become inefficient (over 20 seconds to consider 21 transformations). We conclude that SA is capable to find good solutions in a reasonable amount of time.

7 Conclusion

We proposed an approach to recommend improvements to Web applications. This is based on the idea of leveraging existing work on quality assessment and refactoring. Given a quality model and a set of possible transformations defined with respect to the QM, the approach searches for the best transformations to implement considering cost constraints.

The approach consists of two processes: a suggestion process which uses a meta-heuristic algorithm to propose transformation sequences and a solution evaluation process which uses a QM as a black-box to evaluate these sequences. The approach is general since the exact choice of QM depends on the quality characteristic that needs to be improved. Furthermore, both general and quality-specific transformations can be considered.

We illustrated the approach using a QM evaluating the *navigability* of Web pages. We also presented a set of transformations defined according to their effects on this QM. In a feasibility study, we showed that the proposed approach managed to find the optimal sequence of improvements to 15 different Web pages while being scalable.

It is possible to apply this technique to every page contained in a Web application, but some transformations affect more than one page (e.g. adding a search engine). In future work, we plan on investigating how to propose the best improvements for a whole application instead of treating pages individually.

References

1. Sahraoui, H.A., Godin, R., Miceli, T.: Can Metrics Help to Bridge the Gap between the Improvement of OO Design Quality and its Automation? In: Proceedings of the 16th International Conference on Software Maintenance, pp. 154–162 (2000)
2. Olsina, L., Rossi, G., Garrido, A., Distante, D., Canfora, G.: Incremental Quality Improvement in Web Applications Using Web Model Refactoring. In: Weske, M., Hacid, M.-S., Godart, C. (eds.) WISE Workshops 2007. LNCS, vol. 4832, pp. 411–422. Springer, Heidelberg (2007)
3. Seng, O., Stammel, J., Burkhart, D.: Search-based Determination of Refactorings for Improving the Class Structure of Object-oriented Systems. In: Cattolico, M. (ed.) Proceedings of the 2006 GECCO conference companion on Genetic and evolutionary computation, pp. 1909–1916. ACM, New York (2006)

4. Nielsen, J., Loranger, H.: Prioritizing Web Usability. New Riders, Berkeley (2006)
5. Boldyreff, C., Warren, P., Gaskell, C., Marshall, A.: Web-SEM Project: Establishing Effective Web Site Evaluation Metrics. In: Proceedings of the 2nd International Workshop on Web Site Evolution, Washington, DC, USA, pp. 17–20. IEEE Computer Society, Los Alamitos (2000)
6. Deleuze, C.: Some Points Affecting Web Performance. In: Cordeiro, J.A.M., Pedrosa, V., Encarnação, B., Filipe, J. (eds.) Proceedings of the 2nd International Conference on Web Information Systems and Technologies, pp. 242–245. INSTICC Press (2006)
7. Olsina, L., Lafuente, G., Rossi, G.: Specifying Quality Characteristics and Attributes for Websites. In: Murugesan, S., Desphande, Y. (eds.) Web Engineering. LNCS, vol. 2016, pp. 266–278. Springer, Heidelberg (2001)
8. Albuquerque, A.B., Belchior, A.D.: E-Commerce Websites: a Qualitative Evaluation
9. Malak, G., Sahraoui, H.A., Badri, L., Badri, M.: Modeling Web-Based Applications Quality: A Probabilistic Approach. In: Aberer, K., Peng, Z., Rundensteiner, E.A., Zhang, Y., Li, X. (eds.) WISE 2006. LNCS, vol. 4255, pp. 398–404. Springer, Heidelberg (2006)
10. Caro, A., Calero, C., de Salamanca, J.E., Piattini, M.: Refinement of a Tool to Assess the Data Quality in Web Portals. In: Proceedings of the7th International Conference on Quality Software, pp. 238–243. IEEE Computer Society, Los Alamitos (2007)
11. Du Bois, B., Demeyer, S., Verelst, J.: Does the "Refactor to Understand" Reverse Engineering Pattern Improve Program Comprehension? In: Proceedings of the 9th European Conference on Software Maintenance and Reengineering, Washington, DC, USA, pp. 334–343. IEEE Computer Society, Los Alamitos (2005)
12. Harman, M., Tratt, L.: Pareto Optimal Search Based Refactoring at the Design Level. In: Lipson, H. (ed.) Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, pp. 1106–1113. ACM, New York (2007)
13. Ping, Y., Kontogiannis, K.: Refactoring Web sites to the Controller-Centric Architecture. In: Proceedings of the 8th European Conference on Software Maintenance and Reengineering, pp. 204–213. IEEE Computer Society, Los Alamitos (2004)
14. Ferland, J.A., Costa, D.: Heuristic search methods for combinatorial programming problems. Technical report, Université de Montréal (2001)
15. Haydar, M., Malak, G., Sahraoui, H., Petrenko, A., Boroday, S.: Anomaly Detection and Quality Evaluation of Web Applications. In: Handbook of Research on Web Information Systems Quality. IGI Global (2008)