Expert Systems with Applications 39 (2012) 2270–2278

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A multicriteria ant colony algorithm for generating music playlists

Jose A. Mocholi *, Victor Martinez, Javier Jaen, Alejandro Catala

ISSI – Department of Information Systems and Computation, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

ARTICLE INFO

Keywords:
Ant colony optimization
Music playlist generation
Context-awareness
Ontologies
Semantic search

ABSTRACT

In this paper we address the problem of music playlist generation based on the user-personalized specification of context information. We propose a generic semantic multicriteria ant colony algorithm capable of dealing with domain-specific problems by the use of ontologies. It also employs any associated metadata defined in the search space to feed its solution-building process and considers any restrictions the user may have specified. An example is given of the use of the algorithm for the problem of automatic generation of music playlists, some experimental results are presented and the behavior of the approach is explained in different situations.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

We live in the so-called *information society*, which is a natural consequence of the general advance in technology and especially of the development and growth of the Internet since its birth in the early 90s. Users are now faced with the challenges involved in learning to apply the sources of knowledge generated by the new model of society and to evolve according to its requirements. In dealing with these challenges, they have to handle an information overload that makes it difficult to select the right information and are forced to perform the cumbersome task of exploring excessively dense spaces, an impossible task without the assistance of intuitive and efficient searching tools.

In order to overcome this problem, *recommender systems* were developed in the 90s to facilitate the automatic and personalized selection of the products that best match individual user preferences. They have become an important research area (Adomavicius & Tuzhilin, 2005) since they are problem-rich, have abundant practical applications in helping users to deal with the information overload and provide them with personalized recommendations, content, and services. A good example of these applications is Amazon.com (Linden, Smith, & York, 2003), where customers receive recommendations on books, CDs and other products.

Recommender systems follow two main recommendation strategies: *content-based* and *collaborative filtering*. Content-based filtering relies on descriptions of the items that are being recommended (Melville, Mooney, & Nagarajan, 2001) while collaborative filtering is based on similar past tastes and preferences (Sarwar, Karypis, Konstan, & Reidl, 2001). There are also hybrid approaches that combine collaborative and content-based methods (Adomavicius & Tuzhilin, 2005). However, all these approaches have questionable issues or limitations. When it comes to evaluating the relevance of a set of available products, traditional recommender systems analyze user preferences, which must be properly modeled and stored in a user profile by means of more or less sophisticated mechanisms, none of them free from deficiencies. Generally, the most straightforward approaches simply perform syntactic comparisons against a pre-established set of keywords and hence suffer from the usual shortcomings of mechanisms that do not use semantics to consider the meaning of terms such as synonyms, multiple meanings, etc. Some proposals, based on automatic classifiers, evaluate the relevance of a certain product for a certain user through the use of occurrence patterns of the product attributes or characteristics on a pre-defined training set. Others disregard all product descriptions as a criterion and only consider the associated *rating* (level of interest) previously defined by the user or by other users of the system with similar preferences. For an extended review of common issues and limitations of recommender systems (see Adomavicius & Tuzhilin, 2005).

As new technological breakthroughs become mainstream, the amount of information available to the information society just keeps on growing. This has been estimated as 34 gigabytes for an average person on an average day (Bohn & Short, 2009) only in the USA. In other words, the information overload issue is not getting any better. For example, the amount of recorded music of all types available at any online music store (e.g. iTunes, Amazon.com, etc.) exceeds the average life expectancy. However, as all the music has been annotated with metadata (i.e., valuable knowledge) and the amount of music available will go on growing, we find two requirements that any recommender system should address in this domain: firstly, it has to make use of the available metadata that describe the items and, secondly, it has to be able to deal with large

* Corresponding author. Tel.: +34 96 387 35 69.
E-mail addresses: jmocholi@dsic.upv.es (J.A. Mocholi), vmartinez@dsic.upv.es (V. Martinez), fjaen@dsic.upv.es (J. Jaen), acatala@dsic.upv.es (A. Catala).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.07.131

collections of items, as users tend to add more items to their collections as storage capacity increases. There is also a clear requirement for an effective recommender system: it must be capable of automatically creating a list of recommendations that match the criteria or preferences specified by the user.

In this paper we propose an *ant colony optimization* (ACO) algorithm for use as a recommender system for the automatic generation of music playlists that fulfils the user's predefined set of criteria. It has been designed to cover the two requirements specified above: i.e. it supports multi-criteria recommendation and guarantees good performance when dealing with large sets of items. The paper is organized as follows: in Section 2 we review similar studies. Section 3 gives an overview of ACO, provides a general definition of the problem and describes the functioning of the proposed algorithm. Section 4 presents the domain-specific problem used to test the algorithm, explains the terms of the ontology used and discusses the results obtained from the execution of the algorithm with different sets of criteria. In Section 5 we present our conclusions and describe future work.

2. Related works

In this section we describe approaches from the literature focused on generating music playlists, including both work based on traditional recommender system strategies (content-based and collaborative filtering) that do not allow users to specify arbitrary criteria or constraints and work that does allow this type of specification.

As stated above, content-based filtering techniques look at music attributes such as tempo, pitch and volume and then select songs that conform to these attributes. The idea is that if a user likes a particular song he will also like another similar to it in terms of these attributes. Examples of this approach can be found in Pauws and Eggen (2003), Platt, Burges, Swenson, Weare, and Zheng (2001), and Flexer, Schnitzer, Gasser, and Widmer (2008). Input from the user usually consists of one or more seed songs for which the system then outputs a set of songs. Generally, these systems disregard the multiple metadata associated with the song, so that the generated playlists only contain songs that are very similar in terms of their audio signal features, with the result that these approaches do not bother to sort the songs in the playlist. On the other hand, collaborative filtering is a multi-user approach: it uses other users' explicit preferences to match songs to a specific user. Each user expresses his preferences for a set of songs and the system then tries to expand this set of preferred songs by finding users who have a similar taste and recommending these users' music preferences. In French and Hauver (2001), Hayes and Cunningham (2000) and Kravtsova, Hollemans, Denteneer, and Engel (2002), three collaborative music playlist filtering systems are described that function only on the basis of user likes and dislikes. In a nutshell, both strategies share a common problem whenever the system has to deal with a new user: he has not rated enough songs to create a viable user model to make the recommender system fully effective.

With regard to approaches that allow the specification of constraints, we have to point out the work done in Alghoniemy and Tewfik (2001), a study in which the authors present a network flow approach to playlist generation where each node in the network represents a song, and costs and weights are associated to each arc in order to represent the constraints that have to be satisfied. The goal is to find a continuous path that links the first and last song in the playlist, that has a minimum cost and at the same time has constrained weights. Only two types of constraint are incorporated in the network model: absolute constraints and coherence constraints. The former are a maximum and a minimum percentage of each attribute (e.g. between 60% and 75% of the playlist items should have a certain attribute), and are represented by the weights associated with the arcs in the network model. Coherence constraints, on the other hand, enforce a certain correlation between successive songs in the sequence to ensure that they are similar and are represented by the costs associated with the arcs in the model. Because of the binary song representation, the constraints that can be specified by using these costs and weights of the network are quite simple. To solve the flow problem in playlist generation, it is transformed into an integer linear program which is solved by branch and bound. In the worst case, branch and bound is an exponential algorithm which is in conflict with our scalability requirement. This approach also requires the length of the playlist to be defined in advance by specifying the number of songs to be included in the list, which we believe is an undesirable limitation.

In Pachet, Roy, and Azaly (2000) and Pauws (2002) the authors use constraint satisfaction programming (CSP). The rationale behind this approach is to associate a constrained variable with each position in the playlist, where the domain of each of the variables is the music collection. The idea is to assign a value from the domain to each of the variables, which corresponds to a sequence of songs. In this approach the number of songs in the playlist is fixed by defining the variables in the CSP, which is something users may not want. The solution searching algorithm consists of reducing each domain in turn until it either leads to a solution or fails to find one. Every time a value is removed from a domain, the constraints acting on this domain are inspected to check if any constraint forbids another value belonging to another domain. If this occurs, it is also removed and the process is repeated recursively. This process is known as *constraint propagation*. Domain reduction then occurs either by deliberate choice (assigning a value to a variable), or as a consequence of the propagation required for the satisfaction of certain constraints. Upon failure (a domain becomes empty), the algorithm backtracks to the last decision taken, chooses a different one and tries again. If there is still no solution, the algorithm backtracks one step further and examines the stack of decisions bottom-up until it can either prove that there is no solution or finds one. The solution playlist is therefore constructed by repeatedly adding songs to the playlist and testing whether the playlist can still be completed. We thus only have a partial playlist at our disposal at each iteration, which may be a problem if the algorithm is stopped prematurely. This CSP algorithm has the same worst case complexity as the branch and bound algorithm described above, so again scalability is a concern.

In Aucouturier and Pachet (2002) the authors use the same model as in Pachet et al. (2000) and represent playlist generation as a CSP. However, they use a local search (LS) method, which starts with a random complete assignment of songs to the variables (playlist positions) and iteratively tries to improve upon this playlist by making small changes to it. In order to achieve this behavior, they introduce a cost associated with the playlists: the more constraints that are violated by the playlist and the larger the violation is, the higher the cost. Consequently the best playlist generated is a complete assignment of the variables at each iteration. Since the LS algorithm does not investigate all possible playlists, but rather progressively refines solutions, it also scales up better than the approach defined in Pachet et al. (2000). Another positive aspect is the fact that, instead of choosing a random initial playlist, we are able to use a playlist from a previous iteration of the system as a starting point to guide the algorithm to a final playlist. The downside of this approach is that it represents playlist generation as a constraint satisfaction problem, which means that the number of songs in the playlist is still fixed and also the modifications to the playlist are limited to changing the song that is at a single position in the playlist.

دریافت فوری ← متن کامل مقاله

ISIArticles
مرجع مقالات تخصصی ایران

✓ امکان دانلود نسخه تمام متن مقالات انگلیسی
✓ امکان دانلود نسخه ترجمه شده مقالات
✓ پذیرش سفارش ترجمه تخصصی
✓ امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
✓ امکان دانلود رایگان ۲ صفحه اول هر مقاله
✓ امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
✓ دانلود فوری مقاله پس از پرداخت آنلاین
✓ پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات