Note

h-Assignments of simplicial complexes and reverse search

Sonoko Moriyama\textsuperscript{a}, Masahiro Hachimori\textsuperscript{b}

\textsuperscript{a}Department of Computer Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
\textsuperscript{b}Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan

Received 2 December 2003; received in revised form 20 September 2005; accepted 22 September 2005
Available online 26 October 2005

Abstract

There is currently no efficient algorithm for deciding whether a given simplicial complex is shellable. We propose a practical method that decides shellability of simplicial complexes based on reverse search, which improves an earlier attempt by Moriyama, Nagai and Imai. We also propose to use Macaulay’s theorem during the search. This works efficiently in high-dimensional cases.

Keywords: Shellability; Simplicial complexes; h-Assignments; Reverse search

1. Introduction

Shellability is an important concept which has been extensively studied especially in view of the interplay between topology and combinatorics. However, it is hard to decide whether a given simplicial complex is shellable or not, and it is not known whether this problem can be solved in polynomial time. Moriyama et al. \cite{5} tried a practical method, which they called “the hash-DFS method”, to decide shellability of a simplicial complex \( \mathcal{C} \) using the concept “\( h \)-assignments” introduced in \cite{4}. Their method explores a search space in a depth-first fashion to find a shellable \( h \)-assignment of \( \mathcal{C} \) by using a hash in order to avoid re-searching. It needs, however, to throw some of the data away from the hash as the search proceeds, because the search space is too large while the size of the memory is limited. Here, it uses some heuristic rules for choosing the nodes to be thrown away with hope that the hash holds as many important nodes as possible. Though they verified by experiments that their method works efficiently, the hash-DFS method is naturally incomplete in avoiding re-searching.

In this paper, we propose a framework that uses reverse search \cite{1}. Our method (named “the RS method”) explores the search space \( S \) by traversing a virtual spanning tree \( T(S) \) on \( S \) in a depth-first manner. In this framework, we can theoretically avoid re-searching completely, which was not achieved by the hash-DFS method. Moreover, it uses very little memory because it has only to keep the data of the current node, while the hash-DFS uses a huge hash table.

We further propose to use Macaulay’s theorem during the search which reduces the size of the search space and makes the search more efficient. This works efficiently especially when the dimension of the simplicial complex is high.

\textit{E-mail addresses:} moriso@is.s.u-tokyo.ac.jp (S. Moriyama), hachi@sk.tsukuba.ac.jp (M. Hachimori).
2. Preliminaries and h-assignments

First, we briefly introduce terminology and basic facts needed in this paper. See, for example, [6, Lect. 8] and references therein for the details and further information about shellability.

An (abstract) simplicial complex is a collection \( \mathcal{C} \) of subsets of a finite set \( V = \{1, \ldots, n\} \), faces, such that \( G \subseteq F \in \mathcal{C} \) implies \( G \in \mathcal{C} \). A facet is a maximal face with respect to set inclusion. The dimension \( \dim F \) of a face \( F \) is \(|F| - 1\). The \( n \)-vector of \( \mathcal{C} \) equals to the number of \( n \)-faces in \( \mathcal{C} \).

A simplicial complex is pure if all facets have the same dimension. A ridge of a pure simplicial complex is a \((\dim \mathcal{C} - 1)\)-dimensional face. A ridge contained in only one facet is a boundary ridge.

A d-dimensional pure simplicial complex will be denoted as a d-simplicial complex, for short, in this paper. For a set \( \mathcal{F} \) of faces, \( \overline{\mathcal{F}} \) denotes the simplicial complex generated by \( \mathcal{F} \), i.e., \( \overline{\mathcal{F}} = \{ G : G \subseteq F \in \mathcal{F} \} \).

The f-vector of \( \mathcal{C} \) is \( f(\mathcal{C}) = (f_0(\mathcal{C}), f_1(\mathcal{C}), \ldots, f_d(\mathcal{C})) \), where \( f_i(\mathcal{C}) \) denotes the number of \( i \)-dimensional faces of \( \mathcal{C} \). The h-vector \( h(\mathcal{C}) = (h_0(\mathcal{C}), h_1(\mathcal{C}), \ldots, h_{d+1}(\mathcal{C})) \) is defined by

\[
h_k := \sum_{i=0}^{k} (-1)^{k-i} \binom{d+1-i}{d+1-k} f_{i-1}
\]

\((0 \leq k \leq d + 1)\).

A shelling of a d-simplicial complex \( \mathcal{C} \) is a sequence \( F_1, \ldots, F_t \) of facets such that \( F_i \cap \bigcup_{j=1}^{i-1} F_j \) is a pure \((d+1)\)-simplicial complex for \( 2 \leq i \leq t \). \( \mathcal{C} \) is shellable if it has a shelling. In a shelling \( F_1, F_2, \ldots, F_t \), the restriction set \( \text{Res}(F_j) \) of the facet \( F_j \) is defined by \( \text{Res}(F_j) := \{ v \in F_j : F_j - \{v\} \subseteq F_i \text{ for some } 1 \leq i < j \} \). Here, the number of facets whose restriction set is of the size \( i \) equals to \( h_i(\mathcal{C}) \). See [6, Lect. 8] for details about this fact.

Inspired from the relation between \( n \)-vectors and shellings, an h-assignment is defined as follows [4]. An h-assignment is an assignment of a label \( i \) with \( 0 \leq i \leq d + 1 \) to each facet such that the number of facets with the label \( i \) equals to \( h_i(\mathcal{C}) \). Hence, a simplicial complex \( \mathcal{C} \) is shellable if and only if \( \mathcal{C} \) has a shellable h-assignment. Here, a shellable h-assignment can be characterized as follows.

**Theorem 1 (Moriyama [4]).** Let \( \mathcal{C} \) be a d-simplicial complex and \( A \) its h-assignment. We successively choose an arbitrary facet \( F \) which has \( k \) boundary ridges and whose label is \( d + 1 - k \) in \( A \), for some \( 0 \leq k \leq d + 1 \), and replace \( \mathcal{C} \) by \( \mathcal{C} \setminus F \). Then all facets of \( \mathcal{C} \) can be removed if and only if \( A \) is a shellable h-assignment.

Note that this theorem provides a greedy algorithm to decide whether a given h-assignment is shellable or not.

3. Using reverse search

Theorem 1 shows that all facets of \( \mathcal{C} \) are removed by successive removing steps if the given h-assignment \( A \) is a shellable h-assignment. Here, the order of removing facets corresponds to the reverse order of a shelling. Conversely, we can reconstruct an assignment from the removing order: we label the removed facet by \( d + 1 - \#(\text{boundary ridges in the facet}) \) in each removing step. In other words, if we have an ordering of facets, giving a label to each facet according to this ordering results \( \#(\text{label } i) = h_i \) for each \( i \) if and only if the ordering is a reverse shelling. This suggests checking shellability based on h-assignments as follows. We try to remove the facets one by one and give the labels according to this rule, keeping that the labeling is a partial h-assignment in each step. If all the facets can be removed and labeled, then we get a shellable h-assignment, i.e., the complex is shellable. Otherwise, the complex is not shellable. Note that this strategy is different from checking all possible removing orders according to the definition of shellability, since many different shellings correspond to the same h-assignment. To be precise, this strategy is described as follows.

Let \( \mathcal{C}_A \) be a simplicial complex generated by open facets in a partial h-assignment \( A \). First, we consider an operation, expansion, to assign the label \( d + 1 - k \) to an open facet \( F_i \) in \( A \) when it has \( k \) boundary ridges in \( \mathcal{C}_A \) and the number of facets with the label \( d + 1 - k \) is smaller than \( h_{d+1-k}(\mathcal{C}_A) \). On the other hand, we define the inverse operation, reduction, that removes the label \( k \) assigned to a facet \( F_i \) to get a partial h-assignment \( A' \) when \( F_i \) has \( d + 1 - k \) boundary ridges in \( \mathcal{C}_{A'} \).
Corresponding to these operations, for a simplicial complex $\mathcal{C}$, let us consider a directed graph whose nodes are all partial $h$-assignments and an arc $A \rightarrow B$ is defined if $A$ is converted into $B$ by an expansion. Here we call $A$ and $B$ a parent assignment and a child assignment, respectively. We define $G(S)$ as the subgraph induced by the set $S$ of the nodes which can be reached by a directed path from the empty assignment. $G(S)$ is a rooted acyclic directed graph having the empty assignment as a root.

Now we have only to search $G(S)$ in a depth-first fashion to check whether there exists an $h$-assignment, thus a shellable $h$-assignment, of $\mathcal{C}$. But this method is so simplistic that we would re-search the same partial $h$-assignments repeatedly because $G(S)$ is not a tree. Thus, we propose to use reverse search here. The idea of reverse search is that if we define a unique parent to each node (other than the root) among its parents in the graph $G(S)$, naturally get a spanning tree $T(S)$ of $G(S)$ and we can search on the tree without re-searching. This idea is introduced by Avis and Fukuda [1] which is used in many kinds of combinatorial enumeration problems.

For this aim, we assume the facets of $\mathcal{C}$ are indexed by $1 \leq i \leq m$ in some arbitrarily fixed way. According to this, we define the unique parent of a partial $h$-assignment $A$ to be the one which is derived from $A$ by applying the reduction to the facet with the largest possible index. In other words, the unique parent of $A$ is a parent assignment which is lexicographically first with respect to the ordering “labeled < open”. This relation defines a spanning tree $T(S)$ of $G(S)$ induced by the edges $A \rightarrow B$ such that $A$ is the unique parent of $B$. We do a depth-first search on this tree. To detect edges $A \rightarrow B$ in $T(S)$ at the node $A$, we check each child assignment whether its unique parent is $A$ or not. This procedure will need $O(m^3d^2)$ time, but practically it can be done more efficiently by maintaining lists of extendable/reducible facets because the modification of such lists occurs only locally in each step. Note that during the depth-first search of $T(S)$, what we should keep is essentially just the data of the current node: there is no need of using recursive call or stacking, as well as no hash tables. Thus, the size of space needed for the computation is $O(m)$ (with a fixed $d$).

We remark that this method can also be used for the enumeration of all shellable $h$-assignments if one wants, while the hash-DFS method cannot.

The method we proposed here is implemented by [3] (rsshell, written in C). As Table 1 shows, it runs faster than the implementation of the hash-DFS method by [5] (SHIDE, written in C++) especially when the size of the data is large. During the experimentation, rsshell used about 500 kbytes memory space for each example, while SHIDE used about 60 Mbytes memory space.

4. Further improvement

In the framework of the method discussed above, we start from the empty assignment and try to grow this assignment up to a full $h$-assignment. Because the ordering of assigning the labels corresponds to the reverse order of a shelling when we get the full shellable $h$-assignment, the simplicial complex $\mathcal{C}_A$ generated by the set of open facets of the partial assignment $A$ should be shellable in each step. This suggests us to use the following Macaulay’s theorem as a necessary condition for $\mathcal{C}_A$ to be shellable.

**Theorem 2 (Macaulay’s theorem).** (see, for example, [6]). If $\mathcal{C}$ is a shellable simplicial complex, its $h$-vector is an $M$-sequence, i.e., it satisfies the conditions $h_0(\mathcal{C}) = 1$ and $h_{k-1}(\mathcal{C}) \geq \binom{k}{h_k(\mathcal{C})}$ for $1 \leq k \leq d + 1$, where $\binom{k}{n} = \binom{a_k}{k-1} + \cdots + \binom{a_1-1}{1}$ if the binomial expansion of $n$ is given by $n = \binom{a_k}{k} + \cdots + \binom{a_2}{2} + \binom{a_1}{1}$ with $a_k > \cdots > a_2 > a_1 \geq 0$. 

Table 2
The difference of computation times without M-sequence check (A) and with M-sequence check (B), computed on Pentium4 3.06 GHz, 1 GB Memory (the dimensions of the data are those of the simplicial complexes, not of the polytopes)

<table>
<thead>
<tr>
<th>Data</th>
<th>A (s)</th>
<th>B (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>polytopal (shellable)</td>
<td>1.970</td>
<td>3.050</td>
</tr>
<tr>
<td>3-dim, #facets = 49</td>
<td>84.170</td>
<td>127.170</td>
</tr>
<tr>
<td>polytopal (shellable)</td>
<td>35.720</td>
<td>10.670</td>
</tr>
<tr>
<td>4-dim, #facets = 51</td>
<td>1600.760</td>
<td>2046.510</td>
</tr>
<tr>
<td>polytopal (shellable)</td>
<td>44.990</td>
<td>24.620</td>
</tr>
<tr>
<td>4-dim, #facets = 54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>polytopal (shellable)</td>
<td>4.020</td>
<td>0.230</td>
</tr>
<tr>
<td>5-dim, #facets = 57</td>
<td>3896.160</td>
<td>24.340</td>
</tr>
<tr>
<td>random (nonshellable)</td>
<td>74 255.980</td>
<td>224.700</td>
</tr>
<tr>
<td>6-dim, #facets = 27</td>
<td>130 529.460</td>
<td>365.480</td>
</tr>
</tbody>
</table>

When we check whether the $h$-vector of $C_A$ is an M-sequence or not, the calculation of $\ell^k(h_k)$ will be time consuming. But, because the $h$-vector changes only one entry by $\pm 1$ during the search, the computation to get $\ell^k(h_k)$’s can be done easier if one maintains the list $\{a_k, \ldots, a_1\}$. Because the binomial expansion $\{a_k, \ldots, a_1\}$ of $n$ is the $n$th smallest $k$-set in the r-lex (reverse lexicographic) order as explained in [6], one just needs to compute the next or the previous $k$-set in the r-lex order. (The algorithmic rule to get the next one or the previous one can be written down easily.) Also the computation of binomial coefficients can be done easily if we maintain the list $\left\{(a_{k-1} - 1), \ldots, (a_2 - 1), (a_1 - 1)\right\}$ in each step.

Table 2 shows some computational results on several random simplicial polytopes (generated by using POLYMAKE [2], all of them are shellable), and the random simplicial complexes of Table 1. Again this experimentation is done using rsshell of [3]. As this table shows, in the low-dimensional (3-dim) cases the computation times become larger by doing the M-sequence check, whereas in the higher dimensions the check shortens the computation time drastically. This may be because of the fact that Macaulay’s theorem gives more inequalities when the dimension is larger. It seems that the M-sequence check works efficiently for dimensions $\geq 4$. (Note: the condition $h_0(C) = 1$ is implemented in the case of “no M-sequence check”. The implementation of rsshell and SHIDE in Section 3 also use this.)

Acknowledgement

The authors are grateful to Jörg Rambau who brought up the idea which led us to this work.

References