A Short Cut to Optimal Sequences

Akimasa MORIHATA
Research Institute of Electrical Communication,
Tohoku University
2-1-1 Katahira, Aoba-ku, Sendai, JAPAN
morihata@riec.tohoku.ac.jp

Received 31 March 2010
Revised manuscript received 24 August 2010

Abstract We propose a method of developing efficient programs for finding the optimal sequence, such as the maximum valued one among those that are acceptable. We introduce a method of deriving efficient algorithms from naive enumerate-and-choose-style ones. Our method is based on shortcut fusion, which is a program transformation for eliminating intermediate data structures passed between functions, and a set of auxiliary transformations. As an implementation of our method, we introduce a library for finding optimal sequences. The library consists of proposed transformations, together with functions useful to describe desirable sequences, so that naive enumerate-and-choose-style programs will be automatically improved.

Keywords: Functional Programming, Dynamic Programming, Program Transformation, Shortcut Fusion.

§1 Introduction
Suppose that we are preparing an emergency knapsack, in which we would like to put as many useful items as possible. How can we find the best way of doing this? This is the 0-1 knapsack problem, namely finding the most valuable collection of items among those whose total weight is less than a limit. Given items $x$ and limit $u$, the following recursive function, $knap(x, u)$, specifies the best collection of items, where $w$ yields the weight of each item and $\geq_{\text{value}}$ compares collections by the total values of the items.

* A preliminary report of this work was appeared in 23).
\[
\text{knapsack}([], u) = [] \\
\text{knapsack}(a : x, u) = \text{if } w(a) \leq u \land (a : \text{knapsack}(x, u - w(a))) \geq \text{value} \text{ knapsack}(x, u) \text{ then } a : \text{knapsack}(x, u - w(a)) \text{ else } \text{knapsack}(x, u)
\]

We can calculate its solution by memoizing \(\text{knapsack}\) and thereby using a well-known dynamic programming algorithm.

We often attempt at efficiently finding the best solution. However, it is usually difficult to develop efficient algorithms, because their correctness highly depends on details of problems. For example, if we consider a variant of the 0-1 knapsack problem in which we regard flashlights as more valuable when there are also spare batteries in the knapsack, \(\text{knapsack}\) above no longer specifies the best collection. When we want to exclude collections containing too many items, we cannot naively reuse \(\text{knapsack}\), either.

In this paper, we propose a method of developing efficient programs for finding optimal sequences. We consider naive programs in an enumerate-and-choose style as follows.

\[
\text{knapsack}(x, u) = \text{maximum}_{\leq \text{value}} (\text{lessWeighted}_u (\text{subsequences} x))
\]

This program describes the best collection for the 0-1 knapsack problem by three parts: \text{subsequences}, which enumerates all subsequences (subsets) of the items, \text{lessWeighted}_u, which filters out sequences that are heavier than the limit, and \text{maximum}_{\leq \text{value}}, which chooses the most valuable collection. From such a naive and inefficient program, our method derives an efficient one that enumerates only a small number of sequences. Moreover, our method is also applicable to variants of the problem, such as those mentioned above.

In Sect. 3, we formalize our method of deriving efficient algorithms by shortcut fusion\(^{16,17}\). Given a pair of functions, a producer and a consumer of an intermediate result, shortcut fusion collapses the producer-consumer pair into a function by eliminating the intermediate result. We propose a shortcut fusion law that derives efficient algorithms by fusing a maximization operation with an enumeration of feasible solutions. In order to exploit our shortcut fusion law, we also introduce auxiliary program transformations that derive programs for which our shortcut fusion law is applicable from natural descriptions of optimal sequences.

Our theory is extensible to programs to obtain optimal trees, as we sketch out in Sect. 4. We consider finding optimal binary trees, and demonstrate that our theory can be easily extended to deal with them.

In Sect. 5, we introduce a Haskell library for finding optimal sequences. The library consists of proposed transformations, together with functions useful to describe desirable sequences, so that naive enumerate-and-choose-style programs will be automatically improved. Therefore, users are able to develop efficient programs with little algorithmic insight. The transformation is implemented using RULES pragma\(^{28}\) which is an extension of Glasgow Haskell Compiler.\(^*\)1 The library is available from the author’s website.\(^*\)2


\(^*\)2 http://www.riec.tohoku.ac.jp/~morihata/DPSH.tar.gz