Approximate Counting by Dynamic Programming

Vasileios Vasaitis (0459994)
Friday, March 18th, 2005

Abstract
This proposal describes a project that is going to investigate algorithms for counting knapsack solutions and related sets, with a focus on algorithms that utilise dynamic programming. Both the feasibility of the implementation and the performance characteristics of the algorithms are going to be examined; additional goals pertain to actual algorithmic improvements, with derandomisation being the most ambitious of them.

1 Background

1.1 Approximate counting and sampling

In algorithm design, when devising algorithms for dealing with a particular problem, more often than not one deals with the optimisation problem: that is, the problem in question has got a set of (usually finite) solutions, and one searches for the best solution out of those, according to some criteria, or a relatively good solution, if finding the best is too expensive computationally.

Nevertheless, there are alternative problem classes that one might be interested in. One of them is the counting problem where, given a particular problem, the total number of possible solutions is sought. A closely related problem is the sampling problem where, given a particular problem, one must pick one of its solutions (a sample) at random. These two problem classes are related in the sense that they are generally concerned with the full set of solutions of some problem, but also because, in many cases, it is fairly straightforward to base a solution to one of them on a solution to the other.

Unfortunately, counting and sampling problems are almost always very hard in the computational sense, and cannot be solved in polynomial time. This has led most of the research surrounding them to focus on approximation techniques, instead of exact solutions. For the counting problem, such techniques usually return a solution within a multiplicative error factor $\varepsilon$. For the sampling problem, on the other hand, such techniques return a sample whose distribution is within a total variation distance $\delta$ of the uniform distribution. For any such techniques to be considered polynomial, they need to be polynomial in both the size of their input $n$ (with said input being a description of the problem examined), as well as either $\varepsilon^{-1}$ (for counting) or $\log \delta^{-1}$ (for sampling).
There are a handful of methods for devising approximation algorithms, but the most widely applied is the ubiquitous Markov chain Monte Carlo method [4], or MCMC for short. However, the purpose of this project is to investigate an alternative method for approximate counting, and in particular for counting the solutions to the knapsack problem.

1.2 The knapsack problem and contingency tables

The zero-one knapsack problem is defined as follows: given a knapsack of capacity $b$, where $b$ is a positive integer, and a list of integers $a_j$ ($1 \leq j \leq n$) representing the sizes of $n$ objects that we would like to pack into the knapsack, a feasible packing is a subset of these $n$ objects, whose total size is less than or equal to $b$.

The optimisation problem in the setting described above also associates a value for each object, and consists of finding a packing that maximises the total value of the contents of the knapsack. The counting problem, on the other hand, consists of counting all the feasible packings, regardless of the amount of space used or the value of the objects. These can be as many as $2^n$, or as few as a single packing, the empty one. Similarly, the sampling problem consists of picking a packing randomly from the set of feasible ones.

Actually, besides the zero-one version (which takes its name by the fact that each object can either appear once in the packing or not at all), the knapsack problem has many more variations. There is also the fractional knapsack, where one may pick a fraction of each object (and which is a fairly trivial problem), the general integer knapsack, where there can be multiple instances of each object in the packing, and even the non-integer knapsack, which is particularly difficult to solve even in the optimisation case. In addition, any of the above variations may be extended by requiring that the objects satisfy multiple size constraints at once; this is usually referred to as the multidimensional knapsack.

Another problem related to knapsack is that of contingency tables. Given $m$ row sums and $n$ column sums, a contingency table is a two-dimensional, $m \times n$ matrix of non-negative integers, such that the sums of its rows and columns are equal to the corresponding given sums. Contingency tables have a lot of applications in areas such as statistics, for example with the $\chi^2$ independence test.

1.3 Counting knapsack and contingency tables using dynamic programming

As with most approximate counting and sampling work that has been done so far, most of the research on counting the above problems has focused on MCMC approaches. For the zero-one knapsack problem, the best MCMC result is due to Morris and Sinclair [5, 6], who have proved a mixing time of $O(n^{9/2+\epsilon})$
steps\(^1\), where \(n\) is the number of objects and \(\epsilon > 0\), for a particularly simple and natural Markov chain. For the multidimensional knapsack, again Morris and Sinclair [6] have proved a polynomial mixing time of \(n^{O(m)}\), where \(m\) is the number of dimensions of the problem. Evidently, this is a polynomial result only when the number of dimensions is considered to be a constant.

For contingency tables, the situation is quite similar. In particular, when the number of rows \(m\) is constant, the best result utilising MCMC methods is due to Cryan, Dyer, Goldberg, Jerrum and Martin [1], with a mixing time of about \(n^{O(m)}\). On the other hand, when both the number of rows and columns is variable, a polynomial-time solution has been found by Dyer, Kannan and Mount [3] for cases when the row and column sums are sufficiently large; more specifically, when the row sums are \(\Omega(n^2m)\) and the column sums are \(\Omega(m^2n)\).

Recently however, Dyer [2] developed an algorithm for approximate counting and sampling of the knapsack problem, which is partly deterministic, making use of dynamic programming. This algorithm is both simpler and more efficient compared to previous, MCMC approaches to this problem, so it is worth some further investigation.

For the one-dimensional, zero-one knapsack, Dyer's approach is to scale down the values associated with the problem, i.e. both \(b\) and \(a_j\), by applying the function \(\lfloor n^2/b \rfloor \cdot (\cdot)\). Then the upper bound of the scaled down problem obviously becomes \(n^2\), and counting its possible solutions can be achieved with dynamic programming in \(O(n^3)\) time. He then proceeds to prove that the set of solutions to the scaled down problem is a superset of those of the actual problem, with the number of solutions to the former being at most \(n + 1\) times the number of solutions to the latter.

Having counted the number of solutions to the scaled down problem exactly and deterministically, he then describes an efficient method of sampling from these solutions, using dart throwing on the table produced by the dynamic programming step. Due to the upper bound mentioned above, both the sampling and the counting based on it are fully polynomial. In particular, the counting time is \(O(n^3 + \epsilon^{-2}n^2)\), with \(\epsilon\) being the multiplicative error of the approximation, as usual. This technique is then generalised and extended for the multidimensional knapsack, with a counting time of \(O(n^{2m+1} + \epsilon^{-2}n^{m+1})\), and the general integer knapsack, this time with a counting time of \(O(n^3 + \epsilon^{-2}n^3)\).

Similar ideas are also applied to contingency tables, yielding an approximate counting technique with a running time of \(O(n^4m + 1 + \epsilon^{-2}n^3m)\).

2 The project

The goal of this project is to investigate algorithms for counting knapsack solutions and for counting related sets, such as the number of solutions to contingency tables. Hopefully the project will also result in improving these algorithms, or even devise new ones. Naturally, the main focus is going to be on

\(^{1}\)For more details about MCMC and the terminology surrounding it, the reader is referred to Jerrum's excellent survey on the subject [4].
Dyer's algorithms, since they follow a novel approach, but also because they provide the best results in the area so far.

In more detail, what follows is the breakdown of the various steps of the project, in the order that they are going to be undertaken.

**Implementation** The first step is to produce a working implementation of Dyer's algorithms, so that their actual performance can be measured. First of all, the base algorithm that addresses the zero-one knapsack is going to be implemented. This is going to be followed both by the multidimensional generalisation of that, as well as the related algorithm for contingency tables. The algorithm for the general integer knapsack might be implemented as well if time permits, but it has been deemed to be of lower priority.

For the actual implementation, the plan is to make use of a statically typed functional programming language, and in particular OCaml. Such languages provide powerful abstractions for manipulating data structures, and moreover offer strong guarantees about the correctness of the code. Some initial tests about the suitability of OCaml have already shown quite promising results.

In addition, two approaches are going to be tried for the dynamic programming part of the algorithms. One is the typical, bottom-up iterative style of dynamic programming, which fills up the whole array that stores the results of the algorithm in an incremental, systematic fashion. The other approach, however, is the top-down, lazy-evaluation, recursive style of dynamic programming, which starts the calculation from the target value and only calculates the values that are actually needed. The latter is expected to be a big win in speed in most cases, unless of course the array is already quite dense. Furthermore, it can result in space savings as well, by taking advantage of the fact that most of the times the produced array is going to be very sparse. The plan is to develop an efficient (both time- and space-wise) implementation of sparse arrays, which is going to provide a much more compact representation, by not storing in memory the parts of the array that are not occupied. No concrete idea exists yet as to what form this implementation might have, but the most likely structure seems to be some sort of radix trees, which can achieve the crucial balance between space and time efficiency.

Apart from the general algorithmic interest that such an approach is seen to have, it also gives the possibility of implementing the algorithms in an exact counting setting, which is invaluable for evaluating the accuracy of the approximation.

**Evaluation** The next stage is to run tests with the algorithms and evaluate their efficiency, as well as study their behaviour for various inputs. A comprehensive set of input datasets is going to be used, and multiple parameters are going to be measured for each test. These parameters might be: the speed of execution; the accuracy of the approximation; the density of the produced array; and so on. This should aid in shedding light into the actual performance characteristics of the algorithms; what is more, it should provide hints and in-
sights as to the direction that the theoretical investigation of these algorithms might take.

**Algorithmic investigation** All of the algorithms mentioned so far as candidates for implementation and evaluation have this common feature: they first compute the solution to a slightly “expanded” problem in a deterministic fashion, and then use dart-throwing (a randomised technique) to extract an approximation for the actual problem. The increase in size for this expanded problem is bounded; however, the bound is quite loose, so there is a possibility that it can be improved upon, especially in the multidimensional knapsack and contingency tables algorithms. Therefore, the first target of this stage is to investigate this possibility.

In addition to that, another goal is to eliminate the randomised part of the algorithms, thus converting them into completely deterministic ones. This derandomisation is quite a challenge, especially in the world of approximate counting, where few examples exist. Nonetheless, Dyer’s algorithms are simple and straightforward enough that this might indeed turn out to be possible.

Because of the difficulties inherent to derandomisation attempts, an alternative to that is also possible, should such an attempt prove to be unsuccessful. This alternative consists of implementing the MCMC algorithm by Dyer, Kannan and Mount [3] for contingency tables (the only one that stacks up against Dyer’s algorithms, since it covers a different subclass of problems), and testing its efficiency as well. This algorithm has a resemblance to Dyer’s algorithms, in the sense that it performs a random walk on a slightly expanded problem, with a similar bound on the size of that compared to the unmodified problem. Again, the goal is to test whether that bound can be shown to be even more strict than what is currently known.

### 3 Workplan

What follows is a rough plan regarding when each of the subtasks for this project is going to be started, and how long it is expected to last. The first task to be dealt with is of course the implementation of the algorithms. Some preliminary work on this has already started, but the main effort is going to start right at the beginning of the project, at the start of June, and is expected to be finished by the end of June. In addition, as soon as some parts of the implementation are complete, the evaluation phase is going to start as well, probably in mid June; again, this is expected to last for about a month, winding down by mid July.

The algorithmic investigation is going to start at more or less the same time as the evaluation and run in parallel to that, since it is the most challenging task and thus needs the most time. In addition, some of its goals are not at all certain to produce a successful outcome, therefore having enough time is essential, in order to be able to pinpoint possible dead ends and pursue alternatives if necessary. In particular, in case the derandomisation attempts
do not come into fruition, there should be enough time to focus instead on the alternative algorithm for contingency tables. Nonetheless, all parts of the investigation should have been completed by the end of July.

Finally, the writing of the thesis for the project is expected to begin in mid July, right after the completion of both the implementation and the evaluation. This, too, has been allocated an one month period, and should be finished by mid August.

References


