Siu-Wing Cheng
Chung Keung Poon (Eds.)

LNCS 4041

Algorithmic Aspects in Information and Management

Second International Conference, AAIM 2006
Hong Kong, China, June 2006
Proceedings

Springer

TP301.6-53
A111
2006

Siu-Wing Cheng Chung Keung Poon (Eds.)

Algorithmic Aspects in Information and Management

Second International Conference, AAIM 2006
Hong Kong, China, June 20-22, 2006
Proceedings

Springer

Volume Editors

Siu-Wing Cheng
Hong Kong University of Science and Technology, Department of Computer Science
Clear Water Bay, Hong Kong, China
E-mail: scheng@cs.ust.hk

Chung Keung Poon
City University of Hong Kong, Department of Computer Science
83 Tat Chee Avenue, Kowloon Tong, Hong Kong, China
E-mail: ckpoon@cs.cityu.edu.hk

Library of Congress Control Number: 2006927039

CR Subject Classification (1998): F.2.1-2, E.1, G.1-3, J.1

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-35157-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-35157-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11775096 06/3142 5 4 3 2 1 0

Preface

The papers contained in this volume were presented at the Second International Conference on Algorithmic Aspects in Information and Management (AAIM 2006), held on June 20–22, 2006 at the City University of Hong Kong, Hong Kong, China.

The series of AAIM conferences provides an annual international forum for the communication of research advances on algorithms pertinent to information management and management science. The first conference (AAIM 2005) was held in Xi'an, China and it is planned for the near future that conferences of the series will be held in cities in the Pacific Rim.

This volume contains 34 papers selected from a total of 263 papers submitted from places all over the world: Australia, Canada, China, France, Germany, India, Israel, Italy, Japan, Mexico, Mongolia, Netherlands, New Zealand, Poland, Singapore, South Korea, Sweden, Taiwan, Ukraine, UK and USA. In addition to the selected papers, the volume also contains two papers by the invited speakers, Allan Borodin and Ming-Yang Kao.

We thank all the people who made this meeting possible: the authors who submitted papers, the Program Committee members and external reviewers, the invited speakers, the local organizers, and the sponsors for their effort, advice and support. We also thank EasyChair (www.easychair.org) for providing the free conference software.

April 2006

Siu-Wing Cheng
Chung Keung Poon

Conference Organization

AAIM 2006 was jointly organized by the City University of Hong Kong and the Hong Kong University of Science and Technology.

Program Chairs

Siu-Wing Cheng (Hong Kong U of Science and Technology)
Chung Keung Poon (City U of Hong Kong)

Program Committee

Hee-Kap Ahn (Korean Advanced Institute of Science and Technology)
Takao Asano (Chuo U)
Amotz Bar-Noy (City U of New York)
Hans Bodlaender (U of Utrecht)
Peter Brucker (U of Osnabrueck)
Leizhen Cai (Chinese U of Hong Kong)
Jianer Chen (Texas A&M U)
Marek Chrobak (U of California at Riverside)
Rudolf Fleischer (Fudan U)
Joachim Gudmundsson (National ICT Australia)
Gregory Gutin (Royal Holloway, U of London and U of Haifa)
Wen-Lian Hsu (Academia Sinica, Taiwan)
Giuseppe F. Italiano (U of Rome "Tor Vergata")
Tao Jiang (U of California at Riverside and Tsinghua U)
Tak-Wah Lam (U of Hong Kong)
Xiang-Yang Li (Illinois Institute of Technology)
Peter Bro Miltersen (U of Aarhus)
Pat Morin (Carleton U)
Seffi Naor (Technion and Microsoft Research)
Kirk Pruhs (U of Pittsburgh)
Vijaya Ramachandran (U of Texas at Austin)
Rajeev Raman (Leicester U)
Jiri Sgall (Academy of Sciences of Czech Republic)
Paul Spirakis (U of Patras and CTI Greece)
Wing Kin Sung (National U of Singapore)
Hisao Tamaki (Meiji U)
Jan van Leeuwen (U of Utrecht)
Lusheng Wang (City U of Hong Kong)
Yinfeng Xu (Xi'an Jiaotong U)
Binhai Zhu (Montana State U)

External Reviewers

Sang Won Bae
Rezaul Alam Chowdhury
Xiaotie Deng
Andrew Goldberg
Wen-Liang Hwang
Hyunwoo Jung
Yue-Kuen Kwok
James Kwok
Eric Law
Peter Lennartz
Chi-Jen Lu
Esther Moet
Mart Molle
Andrea Pacifici
Guido Proietti
Chan-Su Shin
Maurizio A. Strangio

Organizing Committee

Matthew Chang (City U of Hong Kong)
Mordecai Golin (Hong Kong U of Science and Technology)
Jinxin Huang (Hong Kong U of Science and Technology)
Xiaohua Jia (City U of Hong Kong)
Hing Fung Ting (U of Hong Kong)
Yajun Wang (Hong Kong U of Science and Technology)

Sponsors

City University of Hong Kong
Hong Kong Pei Hua Education Foundation Limited

¥479.00元

Table of Contents

Further Reflections on a Theory for Basic Algorithms

Allan Borodin

Department of Computer Science
University of Toronto
bor@cs.toronto.edu

1 Introduction

Can we optimally solve $Max2SAT$ in (say) time $(|F| \log |F|)$ where $|F|$ is the length of formula F. Of course, since $Max2SAT$ is *NP*-hard, we can confidently rely on our strongly held belief that no *NP*-hard problem can be solved optimally in polynomial time. But obtaining *unconditional* complexity lower bounds (even linear or near linear bounds) remains the central challenge of complexity theory. In the complementary fields of complexity theory and that of algorithm design and analysis, we ask questions such as "what is the best polynomial time approximation ratio" that can be achieved for $Max2SAT$. The best negative results are derived from the beautiful development of PCP proofs. In terms of obtaining better[1] approximation algorithms, we appeal to a variety of algorithmic techniques, including very basic techniques such as greedy algorithms, dynamic programming (with scaling), divide and conquer, local search and some more technically involved methods such as LP relaxation and randomized rounding, semi-definite programming (see [34] and [30] for an elegant presentation of these randomized methods and the concept of derandomization using conditional expectations). A more refined question might ask "what is the best approximation ratio (for a given problem such as $Max2SAT$) that can be obtained in (say) time $O(n \log n)$" where n is the length of the input in some standard representation of the problem. What algorithmic techniques should we consider if we are constrained to time $O(n \log n)$?

In order to bring some coherence to the "Design and Analysis of Algorithms", most courses and texts will organize much of the content in terms of basic "algorithmic paradigms", such as greedy algorithms, backtracking, dynamic programming, divide and conquer, local search, primal-dual, IP/LP rounding, etc. (but not etc. etc.). To this small set of paradigms, we can add randomization and sometimes very creative ways to utilize and combine these basic algorithmic approaches. Although we seem to be able to intuitively describe these basic classes of algorithms, we (in computer science) rarely attempt to *precisely* define what we mean by such terms as greedy, dynamic programming, etc. Clearly, a precise definition is required if we want to defend statements such as "there is no greedy

[1] For maximization (respectively, minimization) problems we will use approximation ratios ≤ 1 (resp. ≥ 1).

algorithm that provides a good approximation for problem X" or "there is no efficient dynamic programming algorithm for optimally solving problem Y".

In the context of combinatorial search and optimization problems, I will suggest some simple but precise algorithmic models for some basic algorithmic paradigms. This is not a new approach, as there were (for example) a number of important attempts to characterize greedy algorithms (e.g. in terms of matroids [16] and greedoids [25]), and characterizing dynamic programming and branch and bound (e.g. in terms of formal languages [20]). In contrast to the elegant abstraction provided by matroids, we are not attempting to (say) characterize when *the* greedy algorithm is optimal for a set system but rather (similar, for example, to some previous studies for local search [24], "branch and bound algorithms" [12] and IP/LP rounding [4]) we are trying to explore the limitations of basic (simply defined) methods in terms of approximation ratios (or time complexity vs approximation/optimality results).

This talk is based on ideas and results from a number of recent papers. In particular, I will present formulations for greedy and greedy-like algorithms [8], simple dynamic programming and backtracking [10], and basic primal-dual/local ratio algorithms [7]. As will be explained, these models are all based on giving priorities to input items and concern worst case time complexity[2] for search and optimization problems.

2 Priority Algorithms as a Model of Greedy and Greedy-Like Algorithms

With the exception of naive brute force search, greedy algorithms are arguably the simplest approach[3] for solving combinatorial optimization problems. The relation between *the* greedy algorithm for set systems and matroids was formalized by Rado [31] and Edmonds [16] with later extension to greedoids [25]. This early development did not address the use of greedy algorithms to achieve guaranteed approximation ratios but rather focused on the question of understanding when *the* greedy algorithm was optimal. Recently, k-extendible set systems are defined in [29] to help address the issue of when *the* greedy algorithm can provide a good approximation. In [8], we offered a simple model, called *priority algorithms* for greedy algorithms that can be applied in a wide variety of applications not restricted to set systems. We will briefly describe this model and some examples of well known greedy algorithms that are captured by this model. In the next section, we will use priority algorithms as the starting point for some other

[2] We note that the algorithmic models can be applied to any measure of complexity (e.g. space complexity, time vs space, average case or smoothed analysis).

[3] On a conceptual level, local search algorithms are perhaps equally simple. Obviously, simplicity is in the eyes of the beholder and ignoring the complexity of optimally solving an LP relaxation, one can easily argue that IP/LP relaxations are also conceptually very simple. But whatever one's experience and intuition, greedy algorithms are certainly considered to be conceptually simple.

"priority based" paradigms, namely "simple dynamic programming", "simple backtracking" and "simple primal dual" algorithms.

The priority model (and the extensions that will follow) relies on the assumption that we represent an input instance as a set of "locally defined input items", each item represented in some "natural" way. For a scheduling problem (such as interval or job scheduling to maximize profit, makespan minimization, etc.) the choice and representation of an "input item" is usually quite natural and not an issue. Namely, for scheduling problems, an input item is a "job" and each job is represented by the parameters of that job (e.g., the duration, deadline, value, etc. of the job). For other applications, such as graph theory there is a choice of whether to represent the items as edges or as vertices, and having done so there is a choice of how much information to provide within the representation of (say) a vertex. For the well known greedy Kruskal and Prim MST algorithms, the input items are edges, represented by their weights and their end points. For greedy vertex cover approximation algorithms the input items are the vertices, each vertex represented by its weight and its list of adjacent vertices[4]. Similarly, for the CNF-SAT problem, we can think of the clauses as the items or the propositional variables as the items. In the latter case, a variable could be represented by a full description of each clause in which it appears.

Of course there is no one "correct way" to define what is an input item and exactly how much "local information" should be included in the representation of an input item. For example, with regard to the interval selection problem, it seems that the most natural representation would be that each input interval is an item and is represented by a triple (s, f, v) where s (respectively, t and v) is the start (respectively, finish and value) of the interval. But one could also represent the input as an interval graph, or combining these representations by representing each interval I by the tuple (s, f, v, L) where L is a list of the intervals that intersect I. But, of course, this representation or the representation as an interval graph could result in a representation of size $\Omega(n^2)$ for an input instance having n intervals. This would seem to defeat the purpose of greedy algorithms which are utilized because of their efficiency.

After agreeing on the nature of the input representation, we are able to define a priority algorithm as formulated in [8]. The basic idea is that a priority algorithm is a *one-pass* algorithm in which the input is processed one input item at a time. The order or priority in which input items are "considered" is determined by a "local ordering". When an input item is considered, the algorithm makes an irrevocable decision about this item. The nature of a problem usually determines the set of allowable decisions (e.g. accept/reject, schedule on particular machine, etc.). But what is an *allowable ordering* of the input items? We impose the priority condition that if inputs I_j and I_k are in the input sets $\mathcal{I}' \subseteq \mathcal{I}$ and I_j has higher priority than I_k in $\mathcal{I}$ then that priority is maintained in the subset $\mathcal{I}'$. In particular then, any function $f : \mathcal{I} \rightarrow \mathbf{R}$ induces an allowable ordering by ordering input items in (say) non-decreasing (or non-increasing) order of their f

[4] In fact, as discussed in [9], it is usually sufficient to represent a vertex by its list of adjacent edges.

value. Any function (including functions of arbitrarily high complexity or even non computable functions) that takes an input item given by its representation and produces a real number can use this real value as the priority of an item. The only other distinction to be made about the ordering is whether the ordering is *fixed* initially (before any item is considered) or if the ordering is *adaptive* in that the priority of an item can depend on the items previously considered.

In either the fixed or adaptive case, we emphasize that priority algorithms do not impose any explicit complexity limitations as the ordering and decisions being made can be of arbitrary complexity. It is only the "syntactic structure" of a priority algorithm that limits its power. Finally, we view priority algorithms as being "greedy-like" and reserve the term *greedy* for those priority algorithms which make "greedy (irrevocable) decisions" for each input item in the sense of making a decision as if this is the last input item and the decision must minimize/maximize the given objective function[5].

We claim that the priority model seems to capture almost all algorithms that we commonly consider as greedy algorithms. The following are examples of fixed order priority algorithms: Kruskals MST, the maximal matching algorithm for unweighted vertex cover, optimal scheduling of unit profit intervals (ordering intervals according to non-decreasing finishing times), Graham's "online" and LPT greedy approximation algorithms for minimizing makespan on identical machines. We also claim that for the exact $MaxkSAT$ problem (where each clause has exactly k literals), the naive randomized algorithm (independently set each variable to true/false with probability 1/2) can be derandomized to be a (online) priority algorithm (where the input items are the propositional variables represented by the description of the clauses in which they appear). This is similar to exercise 16.6 in [34] which indicates how to turn the naive randomized algorithm for MaxCut (independently place each vertex in S or $\bar{S}$ with probability 1/2) into an (online) greedy algorithm. The following are examples of adaptive order priority algorithms: the H_n approximation greedy set cover algorithm, Prim's MST, Dijkstra's shortest path algorithm (for digraphs where all edge costs are non-negative)[6], Huffman optimal prefix trees[7], various greedy algorithms for weighted vertex cover (see, for example, [13]). It also can be shown that the best (to date) polynomial time computable approximation ratio for the uncapactitated facility location problem is a greedy priority algorithm [27].

[5] This "live for today" definition of greediness may not always make sense in settings where the input items are not "isolated"; for example, in a graph problem, if the items are the vertices, we may already know that a given vertex v is present since it is adjacent to a vertex already considered but v itself has not yet been considered. Our view is that in the context of approximation algorithms the distinction between greedy priority and (non-greedy) priority algorithms is not an essential distinction beyond the historical importance of the term. For optimal algorithms, each decision must be greedy by definition.

[6] Here we view Dijkstra's algorithm as computing the optimal tree from a single source to all other vertices.

[7] Here we consider the input items to be nodes of the prefix tree, with leaf nodes representing the keys to be coded and internal nodes representing subtrees.

The priority algorithm model includes online algorithms (where the ordering of input items can be assumed to be dictated by an adversary) and as in the case of online algorithms one can derive negative results on approximation ratios (called the competitive ratio in the online setting). For example, it can be shown that weighted interval scheduling cannot have a c-approximation priority algorithm for any constant c (see [8]). Other negative results for priority algorithms can be found in [8, 32, 15, 3, 10, 9].

But have we captured everything that one would tend to call greedy? In [17], a 4-approximation algorithm called "greedy" is given for the weighted interval scheduling problem. The algorithm is a one pass algorithm in which rejections are irrevocable but acceptances are revocable. The condition that must be satisfied is that after each interval is considered, the partial solution being constructed is feasible. To incorporate such revocable acceptances, an appropriate extension of the priority model has been introduced and studied in [21].

3 Using Priorities Beyond Priority Algorithms

Obviously (and by design) the priority algorithm framework is a very limited algorithmic model. However, a number of other conceptually simple algorithms can also be viewed in terms of priorities given to input items. We briefly consider the stack model of [7], and the BT model of [10].

3.1 The Stack Model as a Model of Simple Primal Dual Algorithms

One of the most interesting developments in approximation algorithms has been the use of primal dual algorithms as pioneered in [1] and [18]. In many cases, primal dual algorithms can be realized as greedy algorithms (for example, as in the greedy approximation algorithms [23, 19] for the uncapacitated facility location problem) and also can be used to analyze known greedy algorithms (i.e. using the method of dual fitting). Simply stated, in a one pass primal dual algorithm, dual variable are increased until at least one constraint becomes tight. When each constraint corresponds to an input variable, then the fact that a constraint becomes tight can be used to make a decision about the corresponding variable. If our input representation has enough information to determine that a constraint has become tight then we can view the primal dual algorithm as an adaptive priority algorithm. For some applications of the primal dual method, a second clean up phase is also required (e.g. for covering problems to remove items not needed for a feasible solution and for packing problems to insure feasibility). The primal dual framework has been shown to be equivalent to the local ratio method [5, 6]. The local ratio method is used in [7] to motivate a stack model which attempts to model primal dual algorithms where the second clean up phase is a simple "popping" of a stack of items that were pushed onto the stack in the first phase (i.e. when dual constraints became satisfied). To be more precise, in a stack algorithm, the priority framework is used to decide on the order in which to consider input items and an "irrevocable accept/reject decision" is replaced by a decision to push the item onto the stack or else reject it. Once again, as in the