# $R^2$ - Heaps with Suspended Relaxation for Manipulating Priority Queues and a New Algorithm for Reweighting Graphs

by

**Ruth Shrairman**

ISBN: 1-58112- 236-5

**DISSERTATION.COM**

Boca Raton, Florida
USA • 2004

*$R^2$ - Heaps with Suspended Relaxation for Manipulating Priority Queues and a New Algorithm for Reweighting Graphs*



Dissertation.com
Boca Raton, Florida
USA • 2004

ISBN: 1-58112- 236-5

# $R^2$-HEAPS WITH SUSPENDED RELAXATION FOR MANIPULATING PRIORITY QUEUES

AND

# A NEW ALGORITHM FOR REWEIGHTING GRAPHS

by

RUTH SHRAIRMAN

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

1995

Shrairman Ruth (Ph.D., Computer Science)
$R^2$-heaps with Suspended Relaxation for Manipulating Priority Queues
And a New Algorithm for Reweighting Graphs
Thesis directed by Professor Harold N. Gabow

This research is dedicated to two main problems in finding shortest paths in the graphs. The *first* problem is to find shortest paths from an origin to all other vertices in *non-negatively* weighted graph. The *second* problem is the same, except it is allowed that *some edges are negative*. This is a more difficult problem that can be solved by relatively complicated algorithms.

We attack the first problem by introducing a new data structure - Relaxed Heaps, that implements efficiently two main operations critical for the improvement of Dijkstra's shortest path algorithm. *$R^2$-heaps with suspended relaxation* proposed in this research gives the best known *worst-case* time bounds of O(1) for a *decrease_key* operation and O(logn) for a *delete_min* operation. That results in the best *worst-case* running time for Dijkstra's algorithm O(m+nlogn). The new data structure is simple and efficient in practical implementation.
The empirical study with $R^2$-heaps demonstrated strong advantage of its use for Dijkstra's algorithm over the "raw" Dijkstra's without heaps. This advantage is especially dramatic for sparse graphs. $R^2$-heaps can be used in a large number of applications in which set manipulations should be implemented efficiently.

For the problem of finding shortest paths in graphs with some negative edges, we present a new approach of reweighting graphs by first reducing the graph to its "*canonical" form,* which allows to apply an effective algorithm to reweight the graph to one with non-negative edges only and simultaneously to find shortest paths from an origin to all other vertices in the graph. This approach allows to give new algebraic and geometric interpretations of the problem. The experiment with the *Sweeping Algorithm* demonstrated $O(n^2 \log n)$ expected time complexity.

These results open new prospects to improve algorithms for a wide variety of problems including different network optimization problems that use Dijkstra's algorithm as a subroutine, as well as multiple Operations Research and Modeling problems that can be reduced to finding shortest paths on graphs.

DEDICATIONS

*To the memory of my mother, Riva*
*whose boundless love and firm belief in me*
*helped  me to accomplish my goals*

*and*

*To my family,*
*Alexander, Igor, and Daniel*
*for their love and support*

ACKNOWLEDGMENTS

I would like to thank my advisor Hal Gabow for giving me an opportunity to explore some of my ideas, for his tremendous insight into algorithmic design that guided me to chose the right directions and to avoid the wrong ones, for being a great teacher in Algorithms and Data Structures, 'Gabow せんせい - sensei', as Japanese people would say, and for his help to orient me in the new environment of the American system of graduate education.

My thanks to Andrzej Ehrenfeucht for helpful discussions of my approach for reweighting algorithm and for critical suggestions that pushed me to improve my presentation. My thanks also to Richard Byrd for his willingness to discuss the algebraic representation of the reweighting algorithm, and for pointing out the deficiency of my first approach. I am very thankful to Fred Glover who invited me to do joint research at the University of Colorado, Boulder, introduced me to this new world, and supported me in many new undertakings. My great appreciation is to Rafael Hassin for his enthusiastic support of the first ideas of the recombinations on the binary trees. The very special thanks to Professor Emeritus Robert Richtmyer for his reading and questioning my material which has brought me to deeper understanding of many points, and for his constant encouraging to continue my research.

My thanks also go to Bill McKenna for his friendship and moral support, to Maya Apelman for her readiness at any time to answer my questions about English. And last but not least, to Alexander Landau for being the first reader of my material and for fruitful discussions. Finally, I would like to thank all my friends who supported me during these years.

CONTENTS

**Part I. $R^2$-heaps With Suspended Relaxation For Manipulating Priority Queues**

TABLES

Table

FIGURES

# I. $R^2$-HEAPS WITH SUSPENDED RELAXATION FOR MANIPULATING PRIORITY QUEUES

## CHAPTER 1

## INTRODUCTION

### 1.1 Review of previous work

In the last twenty years, there has been an explosive growth in the field of combinatorial algorithms. Good algorithm design has been going hand-in-hand with good data structure design, and efficient interplay of data structure, combinatorics, and graph theory has resulted in the most effective algorithms.

Most of combinatorial algorithms manipulate *dynamic* sets, the sets changing over time, and support a variety of operations on them, like *insert element into*, *delete element from*, *extracting the smallest element from,* and others. The best way to implement dynamic sets depends upon the efficiency of the operations that must be supported.

In the family of data structures, the *priority queues*, or *heaps*, are recognized as a useful abstraction due to the large number of applications in which set

manipulations operations should be implemented. A variety of applications, which directly require using priority queues, include such fundamental problems in graph algorithms as finding shortest paths, minimal cost spanning tree, and other network optimization problems.

The priority queue structures include such obvious structures as the Sorted Linear List, and such sophisticated data structures as Fibonacci Heaps. The term "*heap*" was introduced by J. Williams in 1964 in connection with his heapsort algorithm. His heap structure uses linkless representation of a complete binary tree, storing the root in location 1, and the offspring of the node in location $k$, in locations $2k$ and $2k+1$. A heap is further characterized by the requirement that the key contained in any node must be no larger than the keys of its offspring; a tree with this property is said to *be heap-ordered*. Over the years, the term "*heap*" has acquired more general connotation, and is quite often used as an equivalent to the term *priority queue*. We will follow that convention.

Vuillemin [V78] invented a class of heaps, called *binomial queues* that support all heap operations in O(log n)[1] worst case time, here $n$ is the number of items in the heap. Binomial heaps demonstrate improved complexity for

[1] All logarithms here and further are to the base 2.

*union* operation while keeping already known time bounds for most operations supported by binary heaps [CLR92]. We will give a detailed description of the *binomial queues* in section 1.2.

Brown [B78] studied alternative representations of binomial heaps and developed experimental running time bounds. He showed the binomial queue is not just another data structure, but is the most practical structure for priority queues in many situations.

Fredman and Tarjan [FT87] developed an extension of binomial queues called *Fibonacci heaps*, abbreviated as *F-heaps,* that support *delete-min* and *delete* in $O(\log n)$ amortized time and all other heap operations, in particular *decrease-key* in $O(1)$ amortized time. For situations in which the number of deletions is small compared to the total number of operations, F-heaps are asymptotically faster than binomial queues. Thus F-heaps allow to obtain asymptotically faster algorithms for several well-known network optimization problems where the number of deletions is relatively small. For instance, it speeds up Dijkstra's algorithm for single-source shortest path problem with nonnegative length edges to $O(m + n \log n)$ time, where $n$ is the number of vertices and $m$ is the number of edges in the graph. Using Dijkstra's algorithm as a subroutine results in a corresponding improvement of various other network algorithms.

Fibonacci heaps differ from binomial heaps in that they have a more relaxed structure, allowing work that maintains structure *to be delayed* until it is convenient to perform. That results in improved asymptotic time bound, however, the running time for Fibonacci heaps is *amortized*, not worst-case, time. Amortized analysis is used here to show that in the worst case, an average cost of an operation is small, even though a single operation might be expensive. (Probability is not involved in the amortized analysis.)

Peterson [P87] proposed *Vheaps* , a data structure that works with binary trees and uses a variant heap order than the usual one. It allows rotations within trees in order to balance them after *decrease_key* operation that causes a cut of the updated node. This technique has some similarity with the one used in *F-heaps*, but they are not identical in that while *F-heaps* produce cascading cuts, *Vheaps* produce cascading rotations. It was shown that *Vheaps* are just as asymptotically efficient as F-heaps, giving the same *amortized* time bounds.

In " Introduction to Algorithms" by T. Cormen, C. Leiserson, and R. Rivest [CLR92], the authors state: "From a practical point of view, however, the constant factors and programming complexity of Fibonacci heaps make them less desirable than ordinary binary...heaps for most applications. Thus, Fibonacci heaps are predominantly of theoretical interest. If a much simpler

data structure with the same amortized time bounds as Fibonacci heaps were developed, it would be of great practical use as well."

Driscoll, Gabow, Shrairman, and Tarjan [DGST88] presented another extension of binomial queues, a new data structure called *Relaxed Heaps* which does answer this appeal: it presents an alternative to *Fibonacci heaps* , a much simpler data structure which demonstrates theoretical and practical improvements over the *Fibonacci heaps* .

The Relaxed heaps allow more relaxation than F-heaps: it is allowed not only to delay work to maintain structure, but also *to keep a limited amount of violations in the heap structure.* Furthermore, while the *Fibonacci heaps* use the relaxation of the structural organization itself, the *Relaxed heaps* use the *relaxation in the key relations only* , keeping the structure unchanged. The strict structural discipline of the Relaxed heaps results in the improved complexity for the *worst case* vs. amortized time bound, namely *O(1)* for *decrease_key* and *O(log n)* for *delete_min* operations, and it gives advantage over the *Fibonacci heaps* in practical implementation as well.

Two versions of the *Relaxed heaps* are given. The first, Rank Relaxed Heaps achieves the same *amortized* bounds as *Fibonacci heaps*, but maintains more structure. The second, Run Relaxed Heaps gives theoretical improvement

over *Fibonacci heaps*: it achieves the above time bound for *decrease_key* and *delete_min* in the <u>*worst case*</u> , rather than in the amortized case.

*Relaxed heaps* give also a processor-efficient parallel implementation of the Dijkstra's shortest path algorithm, and hence other algorithms for network optimization [DGST88].

This research presents a further development of *Relaxed heaps*: *Heaps with Suspended Relaxation*, abbreviated as $R^2$*-heaps*. $R^2$*-heaps* combine the strength of both *Rank Relaxed heaps* and *Run Relaxed heaps*: while it keeps the simplicity of the first, it achieves the same *worst case* time bound for *decrease_key* and *delete_min* operations as the second does. That contributes to the name abbreviation $R^2$*-heaps* ("R square heaps").

$R^2$*-heaps* use some new concepts and disciplines in structural organization and manipulation of relaxed heaps that allowed to cut the number of main transformations from six to three, and to decrease the number of pointers per node from 4 to 1 compared to the Relaxed heaps [DGST88]. The experiment run on more than 400 randomly generated graphs in the range of sizes 20 through 600 nodes with different sparseness proved that the $R^2$*-heaps can* be implemented robustly and efficiently. The experiment demonstrated quite

low factors for $m$ and $nlogn$ fractions in the *worst case* time bound for Dijkstra's algorithm[2]: $3m + 22nlogn$ .

## 1.2 Definitions and notations

A *priority queue,* or heap, is a data structure for storing and manipulating a set of items $x$ , each having a numerical key denoted $k(x)$ .

The main operations are:

| | |
|---|---|
| *make_heap* | initialize a heap to store the empty set; |
| *insert(x)* | make $x$ a new item in the heap; |
| *decrease_key* | decrease key $k(x)$ to a smaller value $v$ ; |
| *delete_min* | delete an item of min key from the heap and return it as $x$ ; |
| *find_min* | return an item of minimum key as $x$ ; |
| *delete(x)* | delete item $x$ from the heap. |

A *binomial heap,* or queue, is a collection of heap-ordered binomial trees.

The *binomial trees* $B_r$ are defined recursively as follows:

[2] The factors were determined based on operations count. The formula uses the rounded numbers. The detailed experimental data is presented in Chapter 5.

*$B_0$ is one node; $B_k$ consists of two $B_{k-1}$ trees, the root of one being defined as a child of the root of the other,* see Fig. 1-1. The integer $k$ is called the index, or rank of such a binomial tree.

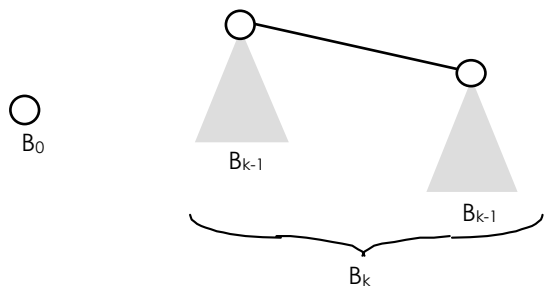


Fig. 1-1 The recursive definition of binomial tree Bk

In all figures, a triangle labeled $r$ represents the *binomial tree* $B_r$. For any k, $0 \le k \le r$, $B_{r+1}$ consists of a $B_k$ tree with additional children of the root that are themselves roots of $B_k, B_{k+1},...,B_r$ trees.

Rank of an arbitrary tree in a binomial queue is the longest path from a leaf of this tree to its root. Rank of $B_0$ is 0. For any node $x$ in a binomial tree, *rank(x)* is the rank $r$ of the maximal subtree $B_r$ rooted at $x$. A binomial tree is an *ordered* tree, with the children of a node ordered by increasing rank. The *last child* of a node is the child of highest rank. The binomial tree $B_r$ has $2^r$ nodes and height $r$. A *binomial queue* for $2^r$ items is a heap-ordered tree $B_r$. A binomial queue for $n$ items, $n$ arbitrary, consists of at most $\lfloor \log n \rfloor + 1$ heap-ordered binomial trees, a tree corresponding each to "one" bit in the binary

expansion of n. An example of the Binomial heap of 11 nodes is presented on Fig. 1-2.

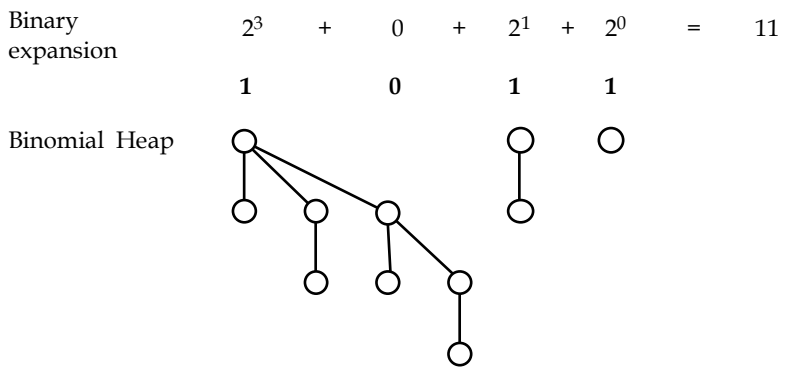


Fig. 1-2 Binomial heap of 11 nodes

It is clear from the definition, that all binomial trees having a given index are isomorphic in the sense that they have the same shape, and all binomial heaps are isomorphic given the total number of nodes.

# CHAPTER 2

# RELAXED HEAPS

## 2.1 Definitions

A *relaxed heap*, abbreviated as R-heap, is a type of binomial queue that allows heap order to be partially violated.

In the tree data structure, each node stores one item, including its key. In our examples, a tree node represents a node of the graph with its label equal to a node key in the tree.

In a *heap-ordered tree* , the parent's key is less or equal to the key of its child:

$$k(p) \leq k(c).$$

If node *c* is the root, or satisfies this relation we call it *good*, otherwise, if $k(p) > k(c)$, it is *bad.* In a heap-ordered structure, all nodes are good. In a *Relaxed heap*, we allow a limited number of violations in the key relations.

If a *decrease_key* operation is applied to a node, this node is considered as *active* , since it represents a potential violation of the structure, and it could be *good* or *bad*. Active node can cease being bad by changing its parent or as a result of its parent key change. The actual status of an active node is checked at the beginning of the appropriate transformation, and if the node is *good*, it is deactivated.

Let $\alpha$ denote the number of *active* nodes in the entire collection of trees at any point of the algorithm. Initially there are no active nodes, so $\alpha$ is zero. The *decrease_key* resets $k(x)$ to $v$ , and each *decrease_key* increases $\alpha$ by one. For each transformation, assume *decrease_key* has created an active node $a$ of rank $r$ with parent $p$ and grandparent $g$ .

One of the main ideas of the Relaxed Heaps is to make *bad* children, that violate the heap order, "play together" in such a way that at least one *bad* child becomes *good*. A variety of group transformations are applied in order to organize this play, and as a result to decrease the number of bad children in $O(1)$ time.

In two implementations of R-heaps, the *Rank Relaxed Heaps (Rank R-heaps)* allow just **one active node per rank**, while the *Run Relaxed Heaps (Run R-heaps)* are less stringent and allow **runs** of *bad* nodes.

We will focus our attention mostly on implementation of *decrease_key* and *delete_min* operations since their efficiency is crucial for improvement of Dijkstra's algorithm asymptotic time bound, and that was our goal and the main motivation in designing the *Relaxed Heaps* . It will be shown that the *Relaxed Heap* transformations allow us to achieve O(m + nlogn) *amortized* time with use of the *Rank R-heaps* and the same, but *worst case* time bound with use of the *Run R-heaps*.

Now we will consider each of these implementations in greater details.

## 2.2 Rank Relaxed Heaps

A *Rank R-heap* is a relaxed binomial queue that satisfies two conditions:

a) For any *r* , there is at most one active node of rank *r* ;

b) Any active node is a last child[3].

A variety of transformations - *pair transformation, good sibling* and *active sibling* transformations, as well as *cleaning* are used in order to support the legal *Rank Relaxed Heap* structure.

We claim that each transformation takes *O(1)* time and either

[3] point *b* is not crucial

i. decrease $\alpha$ , or

ii. does not change $\alpha$ and does not require further transformations to be executed.

For a *Rank R-heap*, if a second active node appears at a given rank, we apply one or a combination of the transformations in order to correct the structure.

We will present several transformations, those that help to illustrate the main ideas used in the Relaxed heaps and those that continue to be used in the next generation of Relaxed Heaps, called ***$R^2$-heaps with suspended relaxation.***

The following notations are used in pictorial illustrations.

An edge joining a child $c$ to its parent $p$ is labeled in four different ways: an arrow from $c$ to $p$ indicates that $c$ is good; cross mark on the edge indicates $c$ is active and bad, dotted edge indicates $c$ may be a new active node; no mark means the status of $c$ is unknown. Each illustration represents a fragment of the heap.

2.2.2 Transformations