REPLICATED HYPERGRAPH PARTITIONING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Reha Oğuz Selvitopi

September, 2010

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Cevdet Aykanat(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Özcan Öztürk

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Oya Ekin Karaşan

Approved for the Institute of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Institute

ABSTRACT

REPLICATED HYPERGRAPH PARTITIONING

Reha Oğuz Selvitopi
M.S. in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
September, 2010

Hypergraph partitioning is recently used in distributed information retrieval (IR) and spatial databases to correctly capture the communication and disk access costs. In the hypergraph models for these areas, the quality of the partitions obtained using hypergraph partitioning can be crucial for the objective of the targeted problem. Replication is a widely used terminology to address different performance issues in distributed IR and database systems. The main motivation behind replication is to improve the performance of the targeted issue at the cost of using more space.

In this work, we focus on replicated hypergraph partitioning schemes that improve the quality of hypergraph partitioning by vertex replication. To this end, we propose a replicated partitioning scheme where replication and partitioning are performed in conjunction. Our approach utilizes successful multilevel and recursive bipartitioning methodologies for hypergraph partitioning. The replication is achieved in the uncoarsening phase of the multilevel methodology by extending the efficient Fiduccia-Mattheyses (FM) iterative improvement heuristic. We call this extended heuristic replicated FM (rFM). The proposed rFM heuristic supports move, replication and unreplication operations on the vertices by introducing new algorithms and vertex states. We show rFM has the same complexity as FM and integrate the proposed replication scheme into the multilevel hypergraph partitioning tool PaToH. We test the proposed replication scheme on realistic datasets and obtain promising results.

Keywords: Hypergraph partitioning, data replication, iterative improvement heuristics.

ÖZET

ÇOKLAMALI HİPERÇİZGE BÖLÜMLEME

Reha Oğuz Selvitopi
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Eylül, 2010

Hiperçizge bölümleme son zamanlarda dağıtık veri erişimi ve uzamsal veri tabanlarında iletişim ve disk erişim maliyetlerini doğru bir sekilde yakalamak için kullanılmıştır. Bu alanlardaki hiperçizge modellerinde, hiperçizge bölümleme kullanılarak elde edilen bölümlerin kalitesi hedeflenen problemin objektifi için çok önemli olabilir. Çoklama, dağıtık veri erişimi ve veri tabanı sistemlerinde çeşitli performans meselelerini ele almak için yaygın olarak kullanılan bir terminolojidir. Çoklamanın arkasındaki ana motivasyon, hedeflenen konunun performansını daha fazla alan kullanma pahasına geliştirmektir.

Bu çalışmada, hiperçizge bölümlemenin kalitesini düğüm çoklamasıyla geliştiren hiperçizge bölümleme şemalarının üstüne odaklanıyoruz. Bu aşamada, çoklama ve bölümlemenin bir arada yapıldığı bir çoklamalı hiperçizge bölümleme şeması öneriyoruz. Yaklaşımımız, hiperçizge bölümlemesi için başarılı çok seviyeli ve özyinelemeli ikiye bölümleme yöntemlerini kullanmaktadır. Çoklama, çok seviyeli yöntemin açılma safhasında verimli Fiduccia-Mattheyses (FM) yinelemeli geliştirme sezgiselini genişleterek elde edilmektedir. Bu genişletilmiş versiyona çoklamalı FM (rFM) diyoruz. Önerilen rFM sezgiseli yeni algoritmalar ve köşe durumları öne sürerek taşıma, çoklama ve azlama işlemlerini desteklemektedir. Önerilen çoklama şemasını çok seviyeli hiperçizge bölümleme aracı PaToH'a entegre edip çeşitli gerçekçi veri takımları üstünde test ediyoruz.

Anahtar sözcükler: Hiperçizge bölümleme, veri çoklama, yinelemeli geliştirme sezgiselleri.

Acknowledgement

I would like to thank to my thesis supervisor Prof. Dr. Cevdet Aykanat for his valuable suggestions, support and guidance throughout the development of this thesis.

I am grateful to Asst. Prof. Dr. Özcan Öztürk and Assoc. Prof. Dr. Oya Ekin Karaşan for reading, commenting and sharing their ideas on the thesis.

I am specially thankful to Ata Türk for sharing his ideas, suggestions and valuable experiences with me throughout the year. I thank Volkan Yazıcı for his comments and support whose endless desire to learn has always inspired me.

I am grateful to my family and their infinite understanding and support throughout my life.

Contents

1 Introduction **1**

2 Background and Problem Definition **4**

2.1 Definitions and Hypergraph Partitioning Problem 4

2.2 Iterative Improvement Heuristics for HP 5

2.3 Multilevel and Recursive Bipartitioning Frameworks 6

2.4 Motivation and Replicated HP Problem 8

3 Related Work **9**

4 Replicated Hypergraph Partitioning **13**

4.1 Replicated FM (rFM) 14

4.1.1 Definitions 14

4.1.2 Overall rFM Algorithm 17

4.1.3 Initial Gain Computation and Gain Update Algorithms . . 20

4.2 rFM and Multilevel Framework 37

4.3 Recursive Bipartitioning and Replica Selection 39

4.3.1 Cut-net Splitting 39

4.3.2 Replica Selection 41

4.3.3 Replication Amount Distribution 43

5 Experimental Results **44**

5.1 Experimental Setup 44

5.2 Datasets 46

5.3 Results 47

6 Conclusion and Future Work **59**

List of Figures

4.1 Move and replication of a vertex. 15

4.2 Unreplication of a replicated vertex. 16

4.3 A net with no non-replicated vertices can be internal to any part. 22

4.4 Various unreplication operations on the replicas of a net that has no non-replicated vertices. 23

4.5 Initial gains and pin distributions. 25

4.6 Gains and pin distributions after moving v_4. 28

4.7 Gains and pin distributions after replicating v_6. 32

4.8 Gains and pin distributions after unreplicating v_1 from V_B 35

4.9 Cut-net splitting, no pins of net n_j are discarded. 40

4.10 Cut-net splitting, the pins of net n_j in part A are discarded. . . . 40

4.11 Three replica selection alternatives for n_j. 42

4.12 Replica selection for n_j ($\lambda_j = 3$) and n_k ($\lambda_k = 2$). 43

5.1 Improvement at $K = 32$. 52

5.2 Improvement at $K = 64$. 53

5.3 Improvement at $K = 128$. 53

5.4 Improvement at $K = 256$. 54

List of Tables

5.1 The properties of the data sets used in our experiments. 46

5.2 The cut and imbalance values for $K = 32$. 48

5.3 The cut and imbalance values for $K = 64$. 49

5.4 The cut and imbalance values for $K = 128$. 50

5.5 The cut and imbalance values for $K = 256$. 51

5.6 The cut values for rFM and gradient rFM for $K = 32$. 55

5.7 The cut values for rFM and gradient rFM for $K = 64$. 56

5.8 The cut values for rFM and gradient rFM for $K = 128$. 57

5.9 The cut values for rFM and gradient rFM for $K = 256$. 58

Chapter 1

Introduction

There are various models that uses hypergraph partitioning for different objectives in different fields such as parallel scientific computing [4, 14, 24, 60], VLSI circuit design [2, 43], distributed IR [13] and database systems [21, 22, 44, 39]. In the areas where hypergraph partitioning (HP) is used, the hypergraph models can broadly be classified into two categories as directional and undirectional hypergraph models. Generally, hypergraph models in parallel scientific computing and VLSI circuit design fall into directional models, whereas hypergraph models in distributed IR and database systems fall into undirectional models.

Recently, undirectional hypergraph models are successfully used to address the issues in distributed information retrieval (IR) [13] and spatial databases [21, 22]. In distributed IR, hypergraph models are used to reduce the communication volume and improve the load balance. In spatial databases, the disk access costs can be reduced for aggregate queries by using hypergraph models. In the hypergraph models for both areas, improving the quality of the partitions obtained using HP by reducing the cutsize is crucial for the problem.

Replication is a widely used terminology in various computer science fields such as distributed IR [5, 11, 47, 48, 50, 57] and database systems [6, 7, 29, 61]. The basic purpose of replication differs from field to field. Generally, replication is used to improve the performance of the target system by reducing the costs

of different objectives at the expense of using more space. Replication is a valuable tool in distributed IR systems to improve the query throughput and fault tolerance. In database systems, the records in the database are replicated across multiple sites to improve the performance of the read operations.

In this work, we propose a replication scheme for hypergraph partitioning that aims to improve the quality of the partitions by replicating vertices that uses undirectional hypergraph models. To our knowledge, this problem has not been addressed for undirectional hypergraph models. In the context of directional hypergraph models, especially in VLSI literature, this is a well-studied problem [40, 33, 34, 45, 63, 26]. In VLSI circuit partitioning, the replication of a vertex in a partitioned circuit may bring internal nets to the cut which are connected to that vertex where input pins of the source vertex need to be replicated along with the replicated vertex since the proposed hypergraph models are directional. In our replication scheme, the replication of a vertex cannot bring any internal net to the cut, i.e., replication cannot increase the cutsize of a bipartition. This forms the basic difference between the replication schemes for directional and undirectional hypergraph models.

Our approach is a single-phase methodology that performs replication along with the partitioning. It uses multilevel and recursive bipartitioning frameworks for HP. We achieve replication in the uncoarsening phase of the multilevel methodology by using a refinement heuristic as a replication tool. We extend the Fiduccia-Mattheyses (FM) heuristic [28] to be capable of replication and unreplication of vertices in addition to standard move operation. We call this extended heuristic replicated FM (rFM). The proposed heuristic operates on a given bipartition and introduces new vertex states and gain update algorithms in order to support replication and unreplication. To obtain multi-way partitions, we adopt recursive bipartitioning methodology. The proposed replication scheme is implemented and integrated into the state–of–the–art HP tool PaToH [15]. In a concurrent work, a two-phase approach [64] is investigated where replication is performed after obtaining a K-way partitioning.

This thesis is organized as follows. Chapter 2 gives the necessary background

for this work and describes replicated HP problem. The previous works regarding replication in various areas are investigated in Chapter 3. Chapter 4 describes our methodology for solving replicated HP problem. We give the experimental results in Chapter 5 and conclude our work in Chapter 6.

Chapter 2

Background and Problem Definition

2.1 Definitions and Hypergraph Partitioning Problem

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$. Each net $n_j \in \mathcal{N}$ connects a subset of vertices. The vertices connected by net n_j are called its *pins*, and denoted as $Pins(n_j)$. The connection between a net n_j and vertex v_i is referred to as a pin (n_j, v_i) of this net. The degree of a net n_j is equal to the number of its pins, $|Pins(n_j)|$. The set of nets that connect vertex v_i is denoted as $Nets(v_i)$ and the size of this set is equal to $|Nets(v_i)|$. Each vertex has a weight associated with it, $w(v_i)$, and each net has a cost value, $c(n_j)$. The cost function for a net is easily extended for a subset of nets $M \subseteq N$ where $c(M) = \sum_{n_j \in M} c(n_j)$.

$\Pi = \{V_1, \ldots, V_K\}$ is a K-way partition of $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ if $V_k \neq \emptyset$ for $1 \leq k \leq K$, and $V_k \cap V_l = \emptyset$ for $1 \leq k < l \leq K$, and $\bigcup_{k=1}^{K} V_k = \mathcal{V}$. A net is said to *connect* a part if it connects at least one pin in that part. *Connectivity set* Λ_i of a net is defined as the set of parts connected by that net. The number of parts in the

connectivity set of n_j is denoted by $\lambda_j = |\Lambda_j|$. A net is said to be *cut* if it connects more than one part ($\lambda_j > 1$), and *uncut* if it connects only one part ($\lambda_j = 1$). The weight $W(V_k)$ of a part V_k is simply the sum of the weights of the vertices in that part.

The *cutsize* of the partition is given by

$$cutsize(\Pi) = \sum_{n_j \in \mathcal{N}} (\lambda_j - 1)c(n_j). \tag{2.1}$$

This is also known as *connectivity* cutsize metric widely used in VLSI [43, 18] and scientific applications [14, 59, 4].

PROBLEM 1. *Hypergraph Partitioning. Given a hypregraph* $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ *and an imbalance value* ϵ*, find a* K*-way partition* $\Pi = \{V_1, \ldots, V_K\}$ *that minimizes the cutsize in Equation 2.1 such that* $(1+\epsilon)W_{avg} \leq W_{max}$ *where,* $W_{avg} = W(\mathcal{V})/K$ *and* $W_{max} = max\{W(V_k)\}$ *for* $k = 1, \ldots, K$*.*

This problem is known to be NP-hard [43].

2.2 Iterative Improvement Heuristics for HP

There are a number of algorithms based on iterative improvement heuristics for solving the HP problem. These heuristics are generally applied to iteratively improve the quality of a random initial partition. An excellent detailed discussion of these techniques can be found in [2]. Most of these algorithms are based on Fiduccia-Mattheyses (FM) [28] and Kernighan-Lin (KL) [38] heuristics, which are designed to improve the cutsize of a bipartition. KL-based heuristics achieve this by swapping vertices from the two parts of the bipartition, while FM-based heuristics use vertex moves from one part to the other. Even though KL-based heuristics perform slightly better in reducing the cutsize, FM-based heuristics are widely used due to their better running-time performance.

In FM-based heuristics, the gain of a vertex is defined as the change in the cutsize of the partition if that vertex were to be moved to its complementary

part in a bipartition. FM heuristics perform multiple *passes* over all vertices, where each pass comprises of a number of iterations. At the beginning of a pass, all vertices are *unlocked*. At each iteration of a pass, the vertex with the highest gain value is moved, *locked* and gain values of its unlocked neighbors are updated. At the end of each iteration, the improvement in the cutsize is stored. A pass terminates when all vertices become locked or there is no feasible move according to balance constraint. At the end of a pass, the bipartition that resulted with the minimum cutsize is restored. Multiple passes can be performed until the improvement in the cutsize drops below a certain threshold. Usually, buckets or heaps are used to store gain values.

The quality of the bipartitions produced by FM heuristic can further be improved at the expense of higher running time. The look-ahead feature and gain vectors are introduced [41] if there are more than one vertex with the highest gain which means a tie-break will occur to select the vertex to move. Different tie-breaking strategies and data structures for gains are investigated [31] and it is shown how the choices can affect the quality of the partitions found by the algorithm. There are also other ways such as using a probabilistic gain computation [25] or adding compaction to FM [55]. On the other hand, FM can be made to run faster [1] with a couple of simple techniques: (i) stop if it is unlikely to make further improvement in a pass; (ii) initialize gains only in the first pass, by rolling back the changes at the end of each pass; and (iii) use only boundary vertices to move, thus reducing the number of vertices to operate on greatly.

2.3 Multilevel and Recursive Bipartitioning Frameworks

KL and FM-based heuristics perform poorly on hypergraphs with high net degrees and they are sensitive to the quality of the initial partition. To alleviate these problems, in 1990s, multilevel algorithms are proposed [10, 32] and successfully applied to HP problem. The basic idea behind the multilevel framework is to perform a sequence of coarsening operations on the original hypergraph to obtain

a coarser hypergraph with small net degrees over which FM heuristics perform particularly better.

Multilevel methodology consists of 3 phases: coarsening, initial partitioning, and uncoarsening. In the *coarsening* phase, $\mathcal{H}_i = (\mathcal{V}_i, \mathcal{N}_i)$ is coarsened into $\mathcal{H}_{i+1} = (\mathcal{V}_{i+1}, \mathcal{N}_{i+1})$ for $i = 0, \ldots, m$. This is achieved by clustering vertices in $\mathcal{H}_i$ where each of these clusters becomes a vertex in $\mathcal{H}_{i+1}$. Starting from the initial hypergraph $\mathcal{H}_0 = (\mathcal{V}_0, \mathcal{N}_0)$, this coarsening operation is iteratively applied to obtain $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ where $|\mathcal{V}_0| > |\mathcal{V}_1| > \ldots > |\mathcal{V}_m|$. Coarsening phase ends when the number of vertices in the coarsest hypergraph drops below a predetermined value. In the *initial partitioning* phase, a bipartition Π_m of the coarsest hypergraph $\mathcal{H}_m$ is obtained. Since the coarsest hypergraph is small, the initial bipartitioning algorithm can be run a couple of times and the best bipartition can be selected. The *uncoarsening* phase contains exactly same number of levels as the coarsening phase. At each level of the uncoarsening, the bipartition Π_i on $\mathcal{H}_i$ is projected back to Π_{i-1} on $\mathcal{H}_{i-1}$. Each vertex in $\mathcal{H}_i$ is decomposed into its forming vertices in $\mathcal{H}_{i-1}$ in the projection. A finer hypergraph will have more degrees of freedom with respect to its coarser counterpart, meaning that the partition can further be improved. This is achieved by the refinement heuristics discussed in Section 2.2.

Recursive Bipartitioning (RB) is the most commonly used method for obtaining K-way partitions of hypergraphs although there are other methods that are based on direct K-way partitioning as in [36, 3]. In the RB paradigm, the initial hypergraph is bipartitioned into two new hypergraphs and then, these two new hypergraphs are further bipartitioned in a recursive manner. This procedure continues until reaching the desired number of parts in $\lg_2 K$ steps. Cut-net splitting scheme [14] is used in order to capture the connectivity cutsize metric in Equation 2.1.

2.4 Motivation and Replicated HP Problem

In this study, we focus on the HP problem with vertex replication. We refer to this problem as replicated hypergraph partitioning problem. Even though a variant of this problem arises in VLSI literature as will be explained in Related Work chapter, our focus is replication in distributed information retrieval and spatial databases. The hypergraph models are used in distributed IR [13] for improving load balance and reducing communication cost. In spatial databases [21, 22], the hypergraph models are used for reducing total disk access cost. In this work, we investigate the effects of replication in the hypergraph models for the mentioned areas where the vertices are replicated. In these areas, replication can help in improving the quality of the partitions by reducing the cutsize at the cost of using more physical space. Although we address the problems in distributed IR and spatial databases, our replication scheme can be used for any area where HP is used as a methodology and the used hypergraph model is undirectional. In our replication scheme, the replication of a vertex should not bring any internal net to the cut as opposed to the replication schemes in VLSI literature. The problem is formulated as follows:

PROBLEM 2. *Replicated Hypergraph Partitioning. Given an* $\mathcal{H} = (\mathcal{V}, \mathcal{N})$*, an imbalance value* ϵ*, and a replication ratio* ρ*, find* K *covering subsets of* $\mathcal{V}$*,* $\Pi^R = \{V_1, \ldots, V_K\}$ *such that* $\bigcup_{k=1}^{K} V_k = \mathcal{V}$*,* $\sum_{k=1}^{K} W(V_k) \leq (1+\rho)W(\mathcal{V})$ *and* $(1+\epsilon)W_{avg} \leq W_{max}$ *where,* $W_{avg} = (1+\rho)W(\mathcal{V})/K$ *and* $W_{max} = max\{W_k\}$ *for* $k = 1, \ldots, K$*.*

Chapter 3

Related Work

Replication is a widely used term in various disciplines of computer science. Specifically, we investigate the replication schemes in VLSI literature, distributed database systems, distributed information retrieval (IR) and spatial databases.

First replication schemes in VLSI circuit design and partitioning arise in the form of gate replication to reduce pin counts and improve the cutsize of the partitioned circuits. In this respect, there are two main approaches in VLSI literature: iterative improvement based replication heuristics that generally use an extended version of the FM heuristic, and graph theoretical approaches that are generally centered on a flow network formulation of the original problem. Replication in logic partitioning requires replication of the pins between the replicated cell (vertex) and its input nets thus, these input nets become cut after replication of that vertex.

One of the first iterative improvement based replication schemes is proposed by Kring and Newton [40]. They provide an extended version of the FM algorithm to handle replication in two-way partitioned networks by introducing new definitions for cell states and moves. There are two different move selection techniques introduced in their work: cleaning of unnecessary replications immediately even if there are higher gain moves and, the prohibition of replications whose gains are under a certain threshold. Kuznar et al. [8] introduced the concept of functional

replication for partitioning a specific FPGA (Field Programmable Gate Array) library, on Xilinx-based devices. Their approach is similar to [40], as they use an FM heuristic by extending it to handle replication. The difference of their work lies in the definition of a cell, which stands for configurable logic blocks rather than logic gates as opposed to other approaches which use FM for VLSI circuit partitioning. In the replication schemes for logic partitioning, when a cell is replicated, all of its input pins must be replicated too. However, in functional replication, some input pins need not to be replicated since they may not be required to generate the replicated cell's required outputs. In this way, they are able to remove some of the nets connected to input pins of the replicated cell which results in better cutsize values. Hwang and El Gamal [33, 34] formulate the min-cut replication problem and give the optimum solution for finding the min-cut replication sets of a K-way partitioned directed graph. Their approach uses a max-flow algorithm to find the replication set of a part that minimizes the cut. They show that this approach can independently be applied to find the replication sets of all parts for a given partition. The drawback of their algorithm is that the size of the replication sets are not guaranteed to be minimum; this may lead to unnecessary replications, and therefore, to the parts' violation of their size limits. To solve the size constrained min-cut replication problem, they first apply the optimal min-cut replication algorithm on each part, and then, if any part violates its size constraint after the replication, a modified FM heuristic is used on those to find a feasible solution. Their solution for hypergraphs is straightforward, which is to replace each net with a directed tree and to use the same algorithm. However, this does not guarantee the minimum cutsize as opposed to partitioning graphs. Liu et al. [45] present an optimal algorithm for the two-way partitioning of graphs with replication and without size constraints. Their formulation of the problem requires a pair of source and sink nodes, rather than an initial partition as in [34]. They use the linear programming method for constructing the replication graph of the original graph, and then, a maximum flow algorithm on this replication graph to find the optimum replication schema. In the case of hypergraphs, they give a heuristic to construct the replication graph and extend FM to a directed FM to partition the replication graph (DFRG) for

minimizing the replication cut cost and satisfying the size constraints. Both approaches, [34] and [45], use a directed version of FM, where a part is chosen to be the source part and only the nets that have a source in the source part and a sink in the other part are considered to be in the cut. [34] uses it on the original graph while [45] uses it on the replication graph. Yang and Wong [63] propose algorithms for finding optimal solutions to the min-area min-cut replication problem for directed graphs and hypergraphs. In their work, they use different flow network models to find min-cut replication sets with minimum sizes. The graph flow network model they use is the same model used in [34]. They propose a new hypergraph flow network model which correctly models the hypergraph. In this approach, their algorithm searches the components in reverse order of [34] which leads to a smaller flow network model. Thus, their algorithm can find smaller cut sizes in a shorter amount of time. Detailed discussion and comparison of replication techniques in circuit partitioning can be found in [26].

In parallel information retrieval systems, the index is partitioned across several machines to be able to process very large text collections efficiently. There are typically two types of index partitioning schemes [57, 35, 12, 49, 51] in distributed IR: document partitioning and term partitioning. Replication is a widely used technique in parallel IR systems to improve query throughput and fault tolerance whatever the partitioning scheme is. Tomasic and Garcia-Molina [58] compare different index distributions in their work and conclude replication is necessary for improving query throughput. Lu and McKinley [47] compare the partial replication and caching to improve the IR system performance and conclude that although caching is simpler and faster, partial replication has the capability of exploiting locality of different queries that require similar answers. They extend their work in [48] by including a study of query locality and a replica selection function. In the distributed IR system of Google [5], the entire system is replicated in order to improve query throughput. Cacheda et al. [11] compare a replicated IR system with a distributed and clustered IR system. Moffat et. al [50] proposes selective inverted list replication to improve the load balancing in a pipelined and term-distributed IR system. Their selective replication approach replicates the inverted lists of high workload terms.

The replication in database systems is used for increasing availability and improving performance of the whole system [6, 7, 29, 61]. The replication is achieved by replicating records in the database across multiple sites. There is a compromise between the consistency of the replicas and the performance of the distributed database system. By replicating a record, the performance of the read operations is increased. However, this performance improvement is bounded by the write operations on the replicated records where serializability and consistency become critical issues as the number of replicas increases. There are mainly two techniques to solve the problem of consistency among multiple replicas. The first one is eager (pessimistic) replication [20, 37] where the write operation on a record is immediately propagated across other replicas and thus, there is one single version of a replica. The other technique is lazy (optimistic) replication [42, 56] where the updates are not propagated immediately and the different versions of the replicas may diverge. Another replication scheme is adaptive replication [62] where the replication of a record may change with respect to read and write patterns on that record. Replication in spatial databases [30, 53] is rather a new and unexplored topic although there are a few studies that explores replication of spatial data [52, 27].

Chapter 4

Replicated Hypergraph Partitioning

The multilevel framework discussed in Chapter 2.3 is enhanced to solve replicated HP problem. Specifically, the replication is achieved in the uncoarsening phase of the multilevel scheme by extending the FM heuristic that is used as the refinement algorithm. We call this extended heuristic replicated FM (rFM). Our proposed algorithm supports move, replication and unreplication operations on the vertices and it strives for minimizing the cutsize while maintaining balance as conventional FM-based algorithms. The multilevel replicated HP algorithm is used in a recursive bipartitioning framework (Section 2.3) to obtain a K-way replicated hypergraph partitioning. In this framework, after each bipartitioning, two sub-hypergraphs are constructed from the obtained partitions and these sub-hypergraphs are used for further bipartitions. In these sub-hypergraphs, the replicated vertices are considered as "real" vertices for the new hypergraphs and necessary pins are added. Thus, the coarsening and the initial partitioning phases of the multilevel scheme can be used as-is without being affected by the replication. After achieving a K-way partition by recursive bipartitioning, in order to compute the cutsize we have to decide which instances of the vertices will be used for each net. This decision affects the cutsize hence, must be done carefully.

4.1 Replicated FM (rFM)

4.1.1 Definitions

In a two-way replicated partition $\Pi^R = \{V_A, V_B\}$, a vertex can belong to V_A, V_B, or to both of them if it is replicated, and hence, it can be in one of three states A, B, AB.

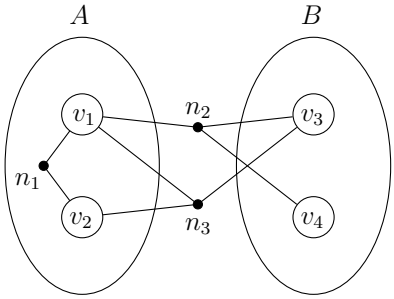
$$State(v_i) = \begin{cases} A & \text{if } v_i \in V_A, \\ B & \text{if } v_i \in V_B, \\ AB & \text{if } v_i \in V_A \text{ and } v_i \in V_B. \end{cases}$$

We use the letters A and B to denote the parts of a bipartition. Each instance of a replicated vertex is referred to as replica.

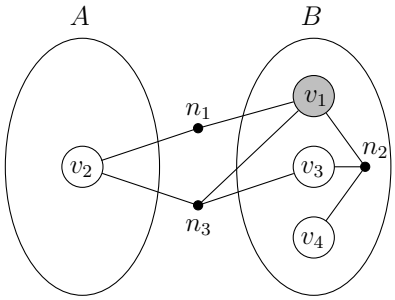
The number of non-replicated vertices that are connected by n_j in A and B are denoted as $\sigma_A(n_j)$ and $\sigma_B(n_j)$ respectively. Similarly, the number of replicated vertices that are connected by n_j is represented by $\sigma_{AB}(n_j)$. In the examples, we use the notation $\sigma(n_j) = (\sigma_A(n_j) : \sigma_B(n_j) : \sigma_{AB}(n_j))$ to denote the pin distribution of n_j. Note that $|Pins(n_j)| = \sigma_A(n_j) + \sigma_B(n_j) + \sigma_{AB}(n_j)$. A net n_j is said to be cut if $\sigma_A(n_j) > 0$ and $\sigma_B(n_j) > 0$. The *cut-state* of a net shows whether the net is cut.

There are 3 operations in rFM: move, replication and unreplication. The move and replication operations are defined for the non-replicated vertices, whereas the unreplication operation is defined for the replicated vertices. Therefore, a non-replicated vertex has move and replication gains whereas a replicated vertex has unreplication gains. Unreplication operation requires two different gain values to be maintained since there are two instances of a replicated vertex in A and B, and each of them may have different unreplication gain. The gain definitions are as follows:

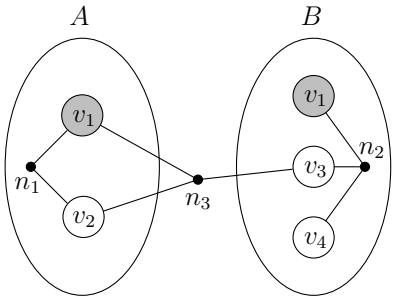
- *Move gain*, $g_m(v_i)$, is defined as the change in the cutsize if the vertex v_i were to be moved to the other part. The move gain of v_i is equal to the difference between the sum of the costs of the nets saved from the cut and



(a) Initial hypergraph before move and replication.



(b) After moving v_1.



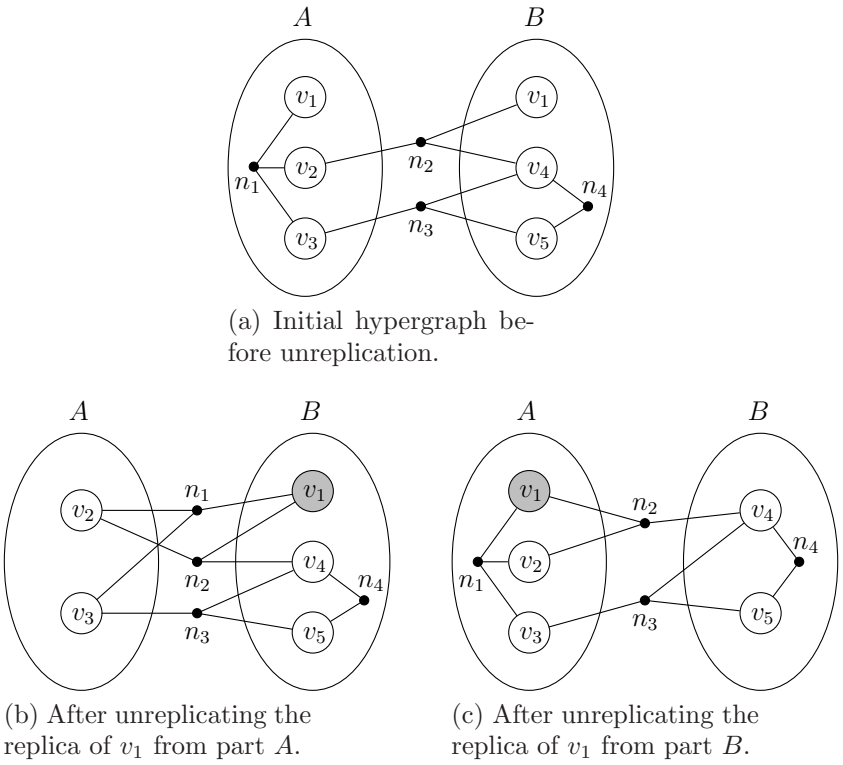
(c) After replicating v_1.

Figure 4.1: Move and replication of a vertex.

the sum of the costs of the internal nets that are brought to the cut. A simple example of move operation is seen in Figs. 4.1a and 4.1b. Moving v_1 from part A to B brings net n_1 into the cut while saving net n_2 from the cut. Hence, $g_m(v_1) = c(n_2) - c(n_1)$. After the move operation, v_1 is locked. The locked vertices in the examples are illustrated by gray color.

- *Replication gain*, $g_r(v_i)$, is defined as the change in the cutsize if the vertex v_i were to be replicated to the other part. The replication gain of v_i is equal to the sum of the costs of the nets saved from the cut. When a vertex is replicated, it cannot bring any internal net to the cut and thus, cannot increase the cutsize. This forms the basic difference between the move and the replication operation. Consequently, for any vertex v_i, we have $g_r(v_i) \geq 0$ and $g_r(v_i) \geq g_m(v_i)$. Figs. 4.1a and 4.1c show the replication of v_1 from part A to B. The replication of v_1 saves net n_2 from the cut as the move of v_1 does, however, net n_1 still remains as an internal net, as opposed to the move operation on the same vertex. Hence, $g_r(v_1) = c(n_2)$. In the

examples, if a net is internal and connects a replicated vertex, we illustrate this by putting a pin to the replica that is in the part of the internal net and omit the pin to other replica. On the contrary, if an external net connects a replicated vertex, a replica of the replicated vertex is randomly chosen to include a pin to that external net.



(a) Initial hypergraph before unreplication.

(b) After unreplicating the replica of v_1 from part A.

(c) After unreplicating the replica of v_1 from part B.

Figure 4.2: Unreplication of a replicated vertex.

- *Unreplication gain*, $g_{u,A}(v_i)$ or $g_{u,B}(v_i)$, is defined as the change in the cutsize if a replica of the replicated vertex v_i were to be unreplicated from its current part. Since unreplication of a replica cannot improve the cutsize, the maximum unreplication gain of a replica is zero. Thus, for any replicated vertex v_i, $g_{u,A}(v_i) \leq 0$ and $g_{u,B}(v_i) \leq 0$. A replica that has an unreplication gain of zero means that this replica is unnecessary and its deletion will not change the cutsize. Furthermore, unreplication of a replica can be "harmful" by bringing internal nets to the cut. This means that the replica is necessary in which case it will have a negative unreplication gain. Fig. 4.2 shows the unreplication of a necessary and an unnecessary replica. Initially,

there are two replicas of v_1 in the sample hypergraph in Fig. 4.2a. The replica in part A is necessary, and its deletion causes the internal net n_1 to be cut, as seen in Fig. 4.2b. On the other hand, the replica in part B is unnecessary, and its deletion does not change the set of nets in the cut, as seen in Fig. 4.2c. Hence, $g_{u,A}(v_1) = -c(n_1)$ and $g_{u,B}(v_1) = 0$.

4.1.2 Overall rFM Algorithm

Replicated FM performs predetermined number of passes over all vertices where each pass comprises of a sequence of operations. Each iteration of a pass starts with selection of a vertex and an operation (move, replication or unreplication) to be performed on that vertex. The selected operation must not violate the size constraints on the weights of the parts. After the selected operation is applied on the vertex, the vertex is locked and the gain values of its unlocked neighbors and the pin distributions of its nets are updated. The size constraints need to be updated if the performed operation is replication or unreplication since the total vertex weight changes. A pass stops when there are no more valid operations. At the end of a pass, a rollback procedure is applied to the point where the partition with the minimum cutsize is seen and then, all vertices are unlocked for further passes. These basic steps are shown in Algorithm 1.

Algorithm 1: Basic Steps of RFM

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), \Pi^R = \{V_A, V_B\}$

Initialize pin distributions, gains and priority queues.
while *there are passes to perform* **do**
    while *there is any valid operation* **do**
        $(v, op) \leftarrow$ Select the vertex and the operation to perform on it with respect to the selection criteria.
        Perform op on v, store the change in the cutsize and lock v.
        Update gains of unlocked neighbors of v and pin distributions of $Nets(v)$.
        if *op is replication or unreplication* **then**
            Update size constraints of parts.
    Unlock all vertices and rollback to the point where the minimum cutsize is seen.

4.1.2.1 Operation Selection

We use a priority based approach for selecting the current operation and disallow some operations that do not satisfy certain conditions. The selection strategy is based on principles such as minimizing the number of unnecessary replicas, limiting replication amount and improving balance.

The highest priority is given to the unreplication operation with a gain value of zero to get rid of unnecessary replica(s) immediately. If a replica has a negative unreplication gain, it means this replica is necessary and its deletion will cause an increase in the cutsize. We do not perform unreplication operations with negative gains and therefore, we do not allow necessary replicas to be deleted.

If there is no replica with an unreplication gain of zero, we make a choice between move and replication by selecting the operation with the higher gain. In the case where the gains of the examined move and replication operations are equal, the move operation is given a higher priority in order to not to consume the given replication allowance. The replication with a gain value of zero is disallowed simply because such operations will produce unnecessary replicas; however, the zero gain moves that improve the balance are performed. Since for any vertex v_i, $g_r(v_i) \geq g_m(v_i)$, in a single pass, the number of replication operations tends to outweigh the number of move operations. This issue can be addressed by the *gradient methodology*, which is discussed in the following subsections.

4.1.2.2 Gradient Methodology

The gradient methodology is proposed and used for the FM heuristics that is capable of replication in [26, 46] to obtain partitions with better cutsize. The basic idea of the gradient methodology is to introduce the replication in the later iterations of a pass, especially when the improvement achieved in the cutsize by performing only move operations drops below a certain threshold. As mentioned in [26], early replication can have a negative effect on the final partition by limiting the algorithm's ability to change the current partition. Furthermore, by using the

replication in the later iterations, the algorithm can proceed by using the local minima reached by the move operations. In rFM, we adopt and modify this methodology by using move and unreplication operations till the improvement in the cutsize drops below a certain threshold and then, we allow replication operations.

4.1.2.3 Early-Exit

We adopt the early-exit scheme for rFM which is a well-known technique to improve the running time performance of the FM-based heuristics. In the early-exit scheme, if there are no improvements in the cutsize for a predetermined number of iterations, the algorithm stops because it is unlikely to further improve the partition.

4.1.2.4 Locking

In conventional move-based FM algorithms, after moving a vertex, it is locked to avoid thrashing. Similarly, in rFM, we also lock the operated vertex after performing a move operation. Furthermore, after performing a replication operation on $v_i \in V_A$, both replicas of v_i are locked. The replica of v_i in part A is locked since the unreplication of this replica would leave the replica in part B alone where performing these two operations (replicating $v_i \in V_A$ and then unreplication of its replica in part A) simply becomes equivalent to performing a move operation on v_i. The unreplication gain of the replica in part B will be negative after the replication of v_i since only positive gain valued replications are allowed which means this replica is necessary and thus, it is locked. Finally, after unreplication of a replicated vertex, the remaining replica, which is now a non-replicated vertex, is locked. That is because the move or replication of this non-replicated vertex will not improve the cutsize.

4.1.2.5 Data Structures

We maintain 6 priority queues keyed according to the gain values of the vertices with respect to operations. For efficiency purposes, the priority queues are implemented as binary heaps. We do not use buckets for storing gain values since the buckets do not perform well when the variation between net costs is high. There are two heaps for the move gains (heapMA and heapMB), two heaps for the replication gains (heapRA and heapRB), and two heaps for the unreplication gains (heapUA and heapUB). A non-replicated vertex has its move and replication gains stored in two heaps (either in heapMA and heapRA or heapMB and heapRB). Similarly, the two replicas of a replicated vertex have their unreplication gains stored in two heaps (in heapUA and heapUB).

In the selection of the vertex and the operation to be performed on that vertex, the root nodes of all heaps are retrieved and the selection is done according to the criteria mentioned above. After the selection is done, we perform an extract-max operation on the heap of the selected vertex and a delete operation on another heap since the selected vertex possesses another gain value. The deletion operation for the other gain value of the selected vertex is required since the vertex is locked and no further operation should be available for this vertex throughout the current pass. For instance, if an extract-max operation is performed on heapMA, it is required to perform a delete operation on heapRA or vice versa. Similarly, if an extract-max operation is performed on heapUA, we need to perform a delete operation on heapUB. Performing an operation may cause gain updates on the neighbors of the operated vertex. A gain update for any vertex may require increase-key or decrease-key operation on the heaps of this vertex belong to.

4.1.3 Initial Gain Computation and Gain Update Algorithms

In this section, we describe the initial gain computation and gain update algorithms in detail. The gain values of the vertices need to be computed at the

beginning of each pass of rFM. After each operation, gain updates may be required for the neighbors of the operated vertex. For each neighbor of the operated vertex, there are at most two gain updates since any vertex possesses two gain values. When compared to the conventional FM-based algorithms in the VLSI literature, rFM has more gain updates and hence, more heap operations. The examples in this section respect to the basics of the operation selection criteria mentioned in Section 4.1.2. For the sake of simplicity, we assume each net has unit cost.

A net n_j is said to be *critical* if an operation on this net would change its cut-state. A net can be critical to a part with respect to its pin distribution. Each type of operation imposes different pin distributions for the criticality of n_j. In the conventional move-based FM algorithm, the move gains are updated whenever the criticality of a net changes. It is interesting to note that the same applies for the update of the replication and the unreplication gains where the criticality of the nets for these two operations is a subset of the criticality of the nets for the move operation. Clearly, the criticality of a net for the move and the replication operation requires at least two non-replicated vertices to exist for that net ($\sigma_A(n_j) + \sigma_B(n_j) > 1$), since the nets that have a single non-replicated pin cannot be critical for the move and replication operations ($\sigma_A(n_j) = 1$ or $\sigma_B(n_j) = 1$). For the move operation,

$$n_j \text{ is move-critical to part } A \text{ if } \sigma_A(n_j) = 1 \text{ or } \sigma_B(n_j) = 0, \tag{4.1}$$

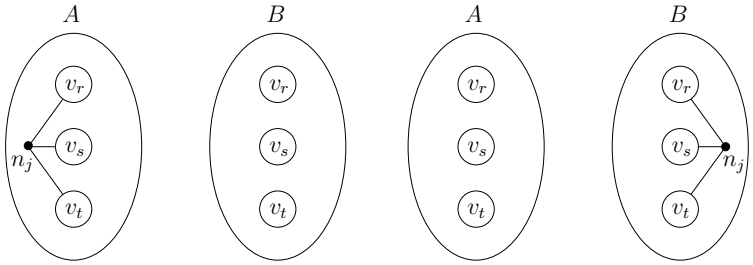
$$n_j \text{ is move-critical to part } B \text{ if } \sigma_B(n_j) = 1 \text{ or } \sigma_A(n_j) = 0. \tag{4.2}$$

The criticality of the internal nets in the move operation is not valid for the replication operation since the replication of a vertex connected to an internal net cannot change the cut-state of that net. However, as in the move operation, the external nets having only one non-replicated pin in a part are critical to that part. Thus, for the replication operation,

$$n_j \text{ is replication-critical to part } A \text{ if } \sigma_A(n_j) = 1, \tag{4.3}$$

$$n_j \text{ is replication-critical to part } B \text{ if } \sigma_B(n_j) = 1. \tag{4.4}$$

For a net to be critical for the unreplication operation, it must have at least one



(a) n_j is internal to part A. (b) n_j is internal to part B.

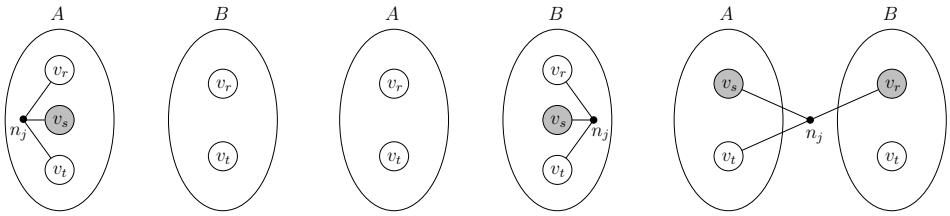
Figure 4.3: A net with no non-replicated vertices can be internal to any part.

replicated vertex ($\sigma_{AB}(n_j) > 0$). As mentioned in Section 4.1.2, the unreplication operation can only change the cut-state of the internal nets. Thus, criticality of a net for the unreplication operation requires it to be internal to a part. Similar to the criticality of the internal nets in the move operation, unreplication operation imposes the same conditions on the criticality of nets. There is an exception for the unreplication operation where a net can be internal but still may not be critical. This exception occurs if the net has pins only to the replicated vertices meaning that the net can be internal to any part, i.e., $\sigma_A(n_j) = \sigma_B(n_j) = 0$ and $\sigma_{AB}(n_j) > 0$. In this condition, unreplication of any replica connected to this net will not make it external. Such nets are also not critical for the move operation since they do not have any non-replicated pins to operate on. Thus, the criticality conditions of the nets for the unreplication operation are still a subset of the criticality conditions of the nets for the move operation. In Fig. 4.3, the net n_j has three replicated vertices v_r, v_s, v_t and no non-replicated vertices. In this case, n_j can be internal to any part as seen in Figs. 4.3a and 4.3b. Unreplication of a replica connected by n_j does not make it external since the number of non-replicated vertices connected by n_j will be equal to one after unreplication and, single pin nets are clearly internal nets as explained. Fig. 4.4 shows various unreplication operations on the bipartition in Fig. 4.3. As seen in Fig. 4.4a, unreplication of the replica of v_s from part B does not make n_j external. Similarly, unreplication of the replica of v_s from part A does not make n_j external in Fig. 4.4b. However, two unreplication operations on n_j, one from part A and one from part B, will make n_j external since $\sigma_A(n_j) > 0$ and $\sigma_B(n_j) > 0$ after these operations. This condition is illustrated in Fig. 4.4c where the replica of v_r

in part A and the replica of v_s in part B are unreplicated. Consequently, for the unreplication operation,

$$n_j \text{ is unreplication-critical to part } A \text{ if } \sigma_B(n_j) = 0 \text{ and } \sigma_A(n_j) > 0, \quad (4.5)$$

$$n_j \text{ is unreplication-critical to part } B \text{ if } \sigma_A(n_j) = 0 \text{ and } \sigma_B(n_j) > 0. \quad (4.6)$$



(a) The bipartition after unreplication of the replica of v_s from part B in the hypergraph in Fig. 4.3.

(b) The bipartition after unreplication of the replica of v_s from part A in the hypergraph in Fig. 4.3.

(c) The bipartition after unreplication of the replica of v_r from part A and unreplication of the replica of v_s from part B in the hypergraph in Fig. 4.3.

Figure 4.4: Various unreplication operations on the replicas of a net that has no non-replicated vertices.

4.1.3.1 Initial gain computation

The initial gain computation given in Algorithm 2 consists of two main loops. The first loop resets the initial gain values by traversing vertices (lines 1–7) and the second loop completes the initialization of gains by traversing pins of the nets (lines 8–20). While resetting gain values, all nets are considered to be internal for the move gains, gainless for the replication gains and external for the unreplication gains. Then, in the completion of the gain values, the move and replication gains are modified for the external and/or critical nets whereas the unreplication gains are modified for the internal (or critical) nets.

The move and replication gains of the non-replicated vertices are initially set to their minimum possible values. If a net n_j is external, its pins' move and replication gains may need to be updated. If this external net is move- or replication-critical to a part (see Equations 4.1, 4.2, 4.3 and 4.4), the move and

Algorithm 2: Initial move, replication and unreplication gain computation

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), \Pi^R = \{V_A, V_B\}$

1 **foreach** $v_i \in \mathcal{V}$ **do**
2 **if** $State(v_i) \neq AB$ **then**
3 $g_m(v_i) \leftarrow -c(Nets(v_i))$
4 $g_r(v_i) \leftarrow 0$
5 **else**
6 $g_{u,A}(v_i) \leftarrow 0$
7 $g_{u,B}(v_i) \leftarrow 0$
8 **foreach** $n_j \in \mathcal{N}$ **do**
9 **foreach** $v_i \in Pins(n_j)$ **do**
10 **if** $State(v_i) \neq AB$ **and** n_j *is external* **then**
11 **if** $(\sigma_A(n_j) = 1$ **and** $State(v_i) = A)$ **or** $(\sigma_B(n_j) = 1$ **and** $State(v_i) = B)$ **then** $\triangleright$ n_j is critical to part A or B
12 $g_m(v_i) \leftarrow g_m(v_i) + 2 * c(n_j)$
13 $g_r(v_i) \leftarrow g_r(v_i) + c(n_j)$
14 **else**
15 $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
16 **else if** $State(v_i) = AB$ **and** n_j *is internal* **then**
17 **if** $\sigma_A(n_j) > 0$ **and** $\sigma_B(n_j) = 0$ **then** $\triangleright$ n_j is critical to part A
18 $g_{u,A}(v_i) \leftarrow g_{u,A}(v_i) - c(n_j)$
19 **else if** $\sigma_B(n_j) > 0$ **and** $\sigma_A(n_j) = 0$ **then** $\triangleright$ n_j is critical to part B
20 $g_{u,B}(v_i) \leftarrow g_{u,B}(v_i) - c(n_j)$

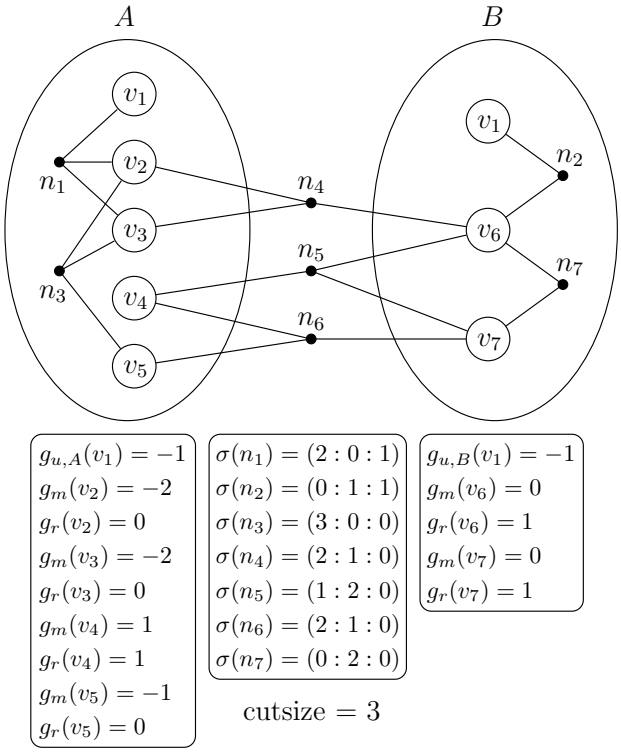


Figure 4.5: Initial gains and pin distributions.

replication gains of the vertex which is in the part that n_j is critical to must be updated since it can be saved from the cut with either move or replication. Its move gain is increased twice $c(n_j)$ since initially all nets are considered to be internal for the move operation and its replication gain is increased by $c(n_j)$. On the other hand, if this external net is not move- or replication-critical, the move gain of its pins are increased by $c(n_j)$. This can be seen as a simple correction for considering all nets as internal while resetting move gain values.

In contrast to move and replication gains, unreplication gains are initially set to their maximum possible values. If a net n_j is unreplication-critical and thus internal (see Equations 4.5 and 4.6), the unreplication gains of its replicated pins may need to be updated. The unreplication gains of the replicas that are in the same part with this internal net need to be decremented by $c(n_j)$ if n_j has at least one non-replicated vertex in the same part.

Fig. 4.5 shows a sample bipartition, the pin distributions of the nets and the

gain values of the vertices in this hypergraph after Algorithm 2 is run. As seen in the figure, the nets n_1, n_3 and n_5 are move-critical to part A, whereas n_2, n_4, n_6 and n_7 are move-critical to part B. The internal nets which are critical for the move operation cannot be critical for the replication operation. If we omit such nets from the set of the critical nets for the move operation, we get the critical nets for the replication operation. Thus, n_5 is replication-critical to part A while n_4 and n_6 are replication-critical to part B. Only internal nets that connects at least one replicated vertex can be critical for the unreplication operation. In this case, n_1 is unreplication-critical to part A and n_2 is unreplication-critical to part B. The nets n_4, n_5 and n_6 are in the cut thus, the cutsize of the bipartition in Fig. 4.5 is 3.

4.1.3.2 Gain updates after move operation

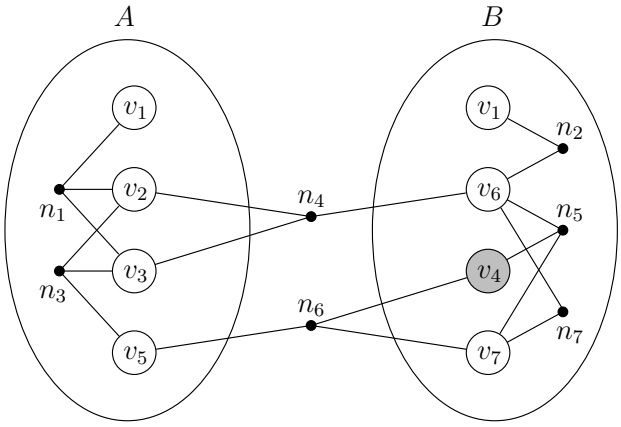
Algorithm 3 shows the procedure of gain updates after moving the given vertex v^* from part A to B. The algorithm includes updating fields of v^*, the pin distributions of $Nets(v^*)$ and the gain values of neighbors of v^*. The necessary field updates on v^* are performed by updating the state and locked fields of v^* to reflect the move operation. The pin distribution of each net $n_j \in Nets(v^*)$ needs to be updated by decrementing $\sigma_A(n_j)$ by 1 and incrementing $\sigma_B(n_j)$ by 1. When the pin distribution of n_j changes, its criticality may change with respect to operation type for part A or B. The change in the criticality of n_j may require various gain updates on the unlocked pins of this net.

After decrementing the number of pins of n_j in A, we check the value of $\sigma_A(n_j)$ to see if the criticality of n_j has changed. If $\sigma_A(n_j) = 0$, n_j becomes internal to part B by becoming move- and unreplication-critical to this part (see Equations 4.2 and 4.6). In this case, the move and the unreplication gains of the unlocked vertices and replicas connected to n_j in B need to be decremented by $c(n_j)$. If $\sigma_A(n_j) = 1$, n_j becomes move- and replication-critical to part A (see Equations 4.1 and 4.3). The only vertex of n_j in A can now save n_j from the cut and thus, the move and replication gains of this vertex must be incremented by $c(n_j)$ if it is unlocked.

Algorithm 3: Gain updates after moving v^* from part A to B

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), \Pi^R = \{V_A, V_B\}, v^* \in V_A$

$State(v^*) \leftarrow B$
Lock v^*
foreach $n_j \in Nets(v^*)$ **do**
    $\sigma_A(n_j) \leftarrow \sigma_A(n_j) - 1$
    if $\sigma_A(n_j) = 0$ **then** $\triangleright$ n_j becomes critical to part B
        foreach *unlocked* $v_i \in Pins(n_j)$ **do**
            if $State(v_i) = B$ **then**
                $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
            else if $State(v_i) = AB$ **then**
                $g_{u,B}(v_i) \leftarrow g_{u,B}(v_i) - c(n_j)$
    else if $\sigma_A(n_j) = 1$ **then** $\triangleright$ n_j becomes critical to part A
        foreach *unlocked* $v_i \in Pins(n_j)$ **do**
            if $State(v_i) = A$ **then**
                $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
                $g_r(v_i) \leftarrow g_r(v_i) + c(n_j)$
    $\sigma_B(n_j) \leftarrow \sigma_B(n_j) + 1$
    if $\sigma_B(n_j) = 1$ **then** $\triangleright$ n_j was critical to part A
        foreach *unlocked* $v_i \in Pins(n_j)$ **do**
            if $State(v_i) = A$ **then**
                $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
            else if $State(v_i) = AB$ **then**
                $g_{u,A}(v_i) \leftarrow g_{u,A}(v_i) + c(n_j)$
    else if $\sigma_B(n_j) = 2$ **then** $\triangleright$ n_j was critical to part B
        foreach *unlocked* $v_i \in Pins(n_j)$ **do**
            if $State(v_i) = B$ **then**
                $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
                $g_r(v_i) \leftarrow g_r(v_i) - c(n_j)$



$g_{u,A}(v_1) = -1$	$\sigma(n_1) = (2:0:1)$	$g_{u,B}(v_1) = -1$
$g_m(v_2) = -2$	$\sigma(n_2) = (0:1:1)$	$g_m(v_6) = -1$
$g_r(v_2) = 0$	$\sigma(n_3) = (3:0:0)$	$g_r(v_6) = 1$
$g_m(v_3) = -2$	$\sigma(n_4) = (2:1:0)$	$g_m(v_7) = -2$
$g_r(v_3) = 0$	$\sigma(n_5) = (0:3:0)$	$g_r(v_7) = 0$
$g_m(v_5) = 0$	$\sigma(n_6) = (1:2:0)$	
$g_r(v_5) = 1$	$\sigma(n_7) = (0:2:0)$	

cutsize = 2

Figure 4.6: Gains and pin distributions after moving v_4.

After incrementing the number of pins of n_j in B, we check the value of $\sigma_B(n_j)$ to see if the criticality of n_j has changed. If $\sigma_B(n_j) = 1$, it means n_j was internal and move- and unreplication-critical to part A (see Equations 4.1 and 4.5). Now because n_j becomes external, the move of the vertices or the unreplication of the replicas connected to n_j in A will not make it external. Thus, the move and the unreplication gains of the unlocked vertices and the replicas in B need to be incremented by $c(n_j)$. Finally, if $\sigma_B(n_j) = 2$, it means n_j was move- and replication-critical to part B (see Equations 4.2 and 4.4). Before the move of v^*, n_j had only one vertex in B which can save n_j from the cut. However, after moving v^* to B, n_j now has two vertices in B and it cannot be saved from the cut anymore. Hence, the move and replication gains of unlocked pin of n_j in B must be decremented by $c(n_j)$.

In Fig. 4.5, when we consider the selection criteria, since there are no replicas with a gain value of zero, we are to select move or replication operation. The highest move gain value is equal to the highest replication gain value which is 1. In such a condition where a tie-break occurs, we select the move operation which

in this case is the move of v_4. Fig. 4.6 shows the bipartition after moving and locking v_4. After updating the pin distributions of $Nets(v_4) = \{n_5, n_6\}$, the gain values of the neighbors of v_4 may need to be updated. The pin distribution of n_5 becomes $\sigma_A(n_5) = 0$ and $\sigma_B(n_5) = 3$. The net n_5 becomes critical to part B and thus, the move gains of the pins of n_5 in B, v_6 and v_7, are decreased by one. The pin distribution of n_6 becomes $\sigma_A(n_6) = 1$ and $\sigma_B(n_6) = 2$. Since $\sigma_A(n_6) = 1$, n_6 becomes critical to part A which means the move and replication gains of v_5 need to be incremented by one. Finally, n_6 was critical to part B before the movement of v_4 meaning that the move and replication gains of v_7 need to be decremented by one. When the gain updates are completed, the cutsize of the bipartition becomes 2. The pin distributions and the gain values after running Algorithm 3 are shown in the table in Fig. 4.6.

4.1.3.3 Gain updates after replication operation

Algorithm 4 shows the procedure of gain updates after replicating the given vertex v^* from A to B. The procedure starts with changing the state of v^* to replicated (AB) and locking both replicas of v^*. Then, for each net n_j connecting v^*, the pin distributions of n_j are updated and checked for their criticality conditions whether they changed or not. Since v^* was in A before replication, $\sigma_A(n_j)$ is decremented by 1 and, $\sigma_{AB}(n_j)$ is incremented by 1 since v^* is replicated now. The replication of v^* from A does not change the $\sigma_B(n_j)$ values of the nets that connect v^*, thus the criticality conditions that include $\sigma_B(n_j)$ need not to be checked.

After decrementing $\sigma_A(n_j)$ for each $n_j \in Nets(v^*)$, we check the pin distribution of n_j regarding $\sigma_A(n_j)$ for the criticality conditions. If $\sigma_A(n_j) = 0$, n_j becomes move- and unreplication-critical for part B (see Equations 4.2 and 4.6). In this condition, the move gains of the unlocked vertices and the unreplication gains of the unlocked replicas connected to n_j need to be decremented by $c(n_j)$ since n_j is internal now and the move of any vertex or the unreplication of any replica would bring it to cut. There are some exceptional cases for the replication and the unreplication operations since the value of $\sigma_B(n_j)$ does not change. One

Algorithm 4: Gain updates after replicating v^* from part A to B

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), \Pi^R = \{V_A, V_B\}, v^* \in V_A$

```
1  State(v*) ← AB
2  Lock v*
3  foreach n_j ∈ Nets(v*) do
4      σ_A(n_j) ← σ_A(n_j) − 1
5      σ_AB(n_j) ← σ_AB(n_j) + 1
6      if σ_A(n_j) = 0 then  ▷ n_j becomes critical to part B
7          foreach unlocked v_i ∈ Pins(n_j) do
8              if State(v_i) = B then
9                  g_m(v_i) ← g_m(v_i) − c(n_j)
10                 if σ_B(n_j) = 1 then
11                     g_r(v_i) ← g_r(v_i) − c(n_j)
12             else if State(v_i) = AB then
13                 if σ_B(n_j) = 0 then
14                     g_{u,A}(v_i) ← g_{u,A}(v_i) + c(n_j)
15                 else if σ_B(n_j) > 0 then
16                     g_{u,B}(v_i) ← g_{u,B}(v_i) − c(n_j)
17     else if σ_A(n_j) = 1 then  ▷ n_j becomes critical to part A
18         foreach unlocked v_i ∈ Pins(n_j) do
19             if State(v_i) = A then
20                 g_m(v_i) ← g_m(v_i) + c(n_j)
21                 if σ_B(n_j) > 0 then
22                     g_r(v_i) ← g_r(v_i) + c(n_j)
```

of these cases occurs for the replication operation if $\sigma_B(n_j) = 1$ where the replication of the only vertex connected to n_j in B will not save n_j from cut anymore and thus, its replication gain must be decremented by $c(n_j)$. The other case is for the unreplication operation and occurs if $\sigma_B(n_j) = 0$ which means there are no non-replicated vertices connected to n_j. In this case, unreplication of the replicas in A were bringing n_j to the cut before the replication of v^*. However, after replication of v^*, unreplication of these replicas will not bring n_j to the cut and thus, their unreplication gains must be incremented by $c(n_j)$.

If $\sigma_A(n_j) = 1$, n_j becomes move- and replication-critical to part A (see Equations 4.1 and 4.3). The move or the replication of the only non-replicated vertex v_i connected to n_j in A can now save n_j from the cut and thus, the move and replication gain of this vertex must be incremented by $c(n_j)$. However, the replication gain of v_i needs to be incremented only if $\sigma_B(n_j) > 0$. That is because in the condition where $\sigma_B(n_j) = 0$ before the replication of v^*, the replication of v_i will not change the cutsize and, after the replication of v^*, the replication of v_i will still not change the cutsize of the partition. Thus, if $\sigma_B(n_j) = 0$, the replication gain of v_i need not be incremented.

After moving v_4, now we are to select another vertex to operate on in Fig. 4.6. There are two operations with the highest gain which are the replication of v_5 and the replication of v_6 and the gain values of these operations are 1. We select to replicate v_6. Fig. 4.7 shows the bipartition after replicating v_6 and locking both replicas of it. After the pin distributions of $Nets(v_6) = \{n_2, n_4, n_5, n_7\}$ are updated, the criticalities of n_2, n_4 and n_7 change and the gain values of the vertices connected to these nets may need to be updated. For n_2, $\sigma_B(n_2) = 0$ which makes it critical to part A and since $\sigma_A(n_2) = 0$, it has no non-replicated vertices. Thus, the unreplication gain of the replica of v_1 in B is incremented by one. The pin distribution of n_4 for B becomes $\sigma_B(n_4) = 0$ making it critical to part A. The move gains of the vertices connected to n_4 in A, v_2 and v_3, need to be decremented by one, however, since $\sigma_A(n_4) = 2$, the replication gain values of these two vertices need not to be updated. Finally, n_7 becomes critical to part B since $\sigma_B(n_7) = 1$ and thus the move and replication gains of the only non-replicated vertex in B that is connected to n_7 may need to be updated. This

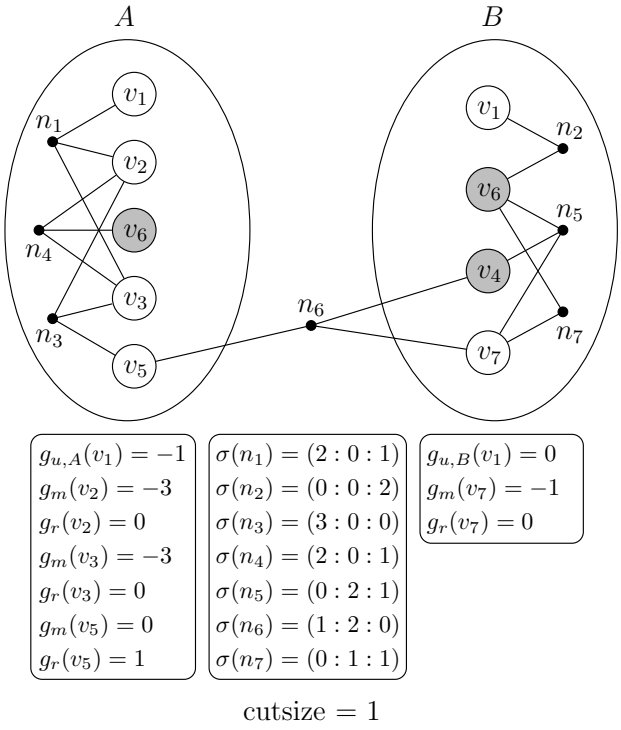


Figure 4.7: Gains and pin distributions after replicating v_6.

vertex is v_7 and its move gain is incremented by one, however, its replication gain is not incremented since $\sigma_A(n_j) = 0$. After the gain updates, the cutsize of the bipartition becomes 1. The gain values and the pin distributions after running Algorithm 4 are shown in Fig. 4.7.

4.1.3.4 Gain updates after unreplication operation

Algorithm 5 shows the procedure of gain updates after unreplication of the given replica v^* from A. Firstly, the state of the replicated vertex v^* is changed to B and it is locked. The pin distributions of each net $n_j \in Nets(v^*)$ are updated by incrementing $\sigma_B(n_j)$ by 1 and decrementing $\sigma_{AB}(n_j)$ by 1. Then, the criticality conditions of the nets connected to v^* are checked for the gain updates of the neighbors of v^*. Since the value of $\sigma_A(n_j)$ does not change, it is not necessary to check the criticality conditions that include $\sigma_A(n_j)$.

After the value of $\sigma_B(n_j)$ is incremented, the criticality of n_j must be checked

Algorithm 5: Gain updates after unreplicating v^* from part A

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), \Pi^R = \{V_A, V_B\}, v^* \in V_A$

$State(v^*) \leftarrow B$
Lock v^*
foreach $n_j \in Nets(v^*)$ **do**
    $\sigma_B(n_j) \leftarrow \sigma_B(n_j) + 1$
    $\sigma_{AB}(n_j) \leftarrow \sigma_{AB}(n_j) - 1$
    if $\sigma_B(n_j) = 1$ **then** $\triangleright$ n_j was critical to part A
        foreach *unlocked* $v_i \in Pins(n_j)$ **do**
            if $State(v_i) = A$ **then**
                $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
                if $\sigma_A(n_j) = 1$ **then**
                    $g_r(v_i) \leftarrow g_r(v_i) + c(n_j)$
            else if $State(v_i) = AB$ **then**
                if $\sigma_A(n_j) = 0$ **then**
                    $g_{u,B}(v_i) \leftarrow g_{u,B}(v_i) - c(n_j)$
                else if $\sigma_A(n_j) > 0$ **then**
                    $g_{u,A}(v_i) \leftarrow g_{u,A}(v_i) + c(n_j)$
    else if $\sigma_B(n_j) = 2$ **then** $\triangleright$ n_j was critical to part B
        foreach *unlocked* $v_i \in Pins(n_j)$ **do**
            if $State(v_i) = B$ **then**
                $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
                if $\sigma_A(n_j) > 0$ **then**
                    $g_r(v_i) \leftarrow g_r(v_i) - c(n_j)$

to see if there are any necessary gain updates for the neighbors of v^*. If $\sigma_B(n_j) = 1$, it means n_j was move- and unreplication-critical to part A (see Equations 4.1 and 4.5). In this case, the move and the replication gains of the unlocked vertices and the unlocked replicas in A are increased by $c(n_j)$ since n_j is no more an internal net. Similar to the gain updates after replication of a vertex, there are two exceptional cases. The first case occurs for the replication operation if $\sigma_A(n_j) = 1$ where the replication gain of the only vertex of n_j in A needs to be incremented since this vertex can save n_j from cut. The other case is for the unreplication operation and occurs when $\sigma_A(n_j) = 0$ meaning that n_j had no non-replicated vertex before the unreplication of v^* and unreplication of the replicas in B were not bringing n_j to the cut. However, after the unreplication of v^*, the unreplication of the replicas in B will bring n_j to the cut and thus, the unreplication gains of these replicas must be decremented by $c(n_j)$.

If $\sigma_B(n_j) = 2$, it means n_j was move- and replication-critical to part B (see Equations 4.2 and 4.4). In this case, n_j has two pins in B and one of them, v^*, is already locked. The move and replication gains of the other vertex, v_i, need to be decremented by $c(n_j)$ since this vertex can no more save n_j from cut. However, if $\sigma_A(n_j) = 0$, it is not necessary to decrement the replication gain of v_i. That is because the replication of v_i does not change the cutsize before or after the unreplication of v^*.

In Fig. 4.7 after the replication of v_6, there exists an unnecessary replica in part B with an unreplication gain of zero. According to the selection criteria, the unreplication operation with a gain of zero has the highest priority. Thus, the selected operation is the unreplication of an unnecessary replica which is the unreplication of v_1 from B. Fig. 4.8 shows the bipartition after the unreplication of v_1 from B and locking it. After the pin distributions of $Nets(v_1) = \{n_1, n_2\}$ are updated, the gains of neighbors of v_1 may need to be updated. The pin distribution of n_1 for A is $\sigma_A(n_1) = 3$ and its criticality has not changed. On the other hand, the pin distribution of n_2 for A is $\sigma_A(n_2) = 1$ which means n_2 was critical to part A. However, since none of the vertices connected to n_2 is unlocked, there is no need to perform gain updates for the pins of n_2. The unreplication of an unnecessary replica cannot change the cutsize thus, after the gain updates,

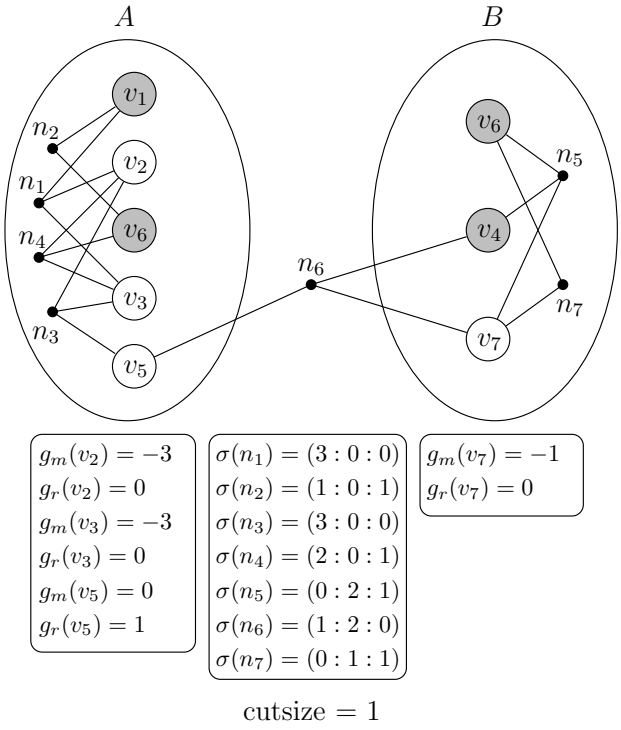


Figure 4.8: Gains and pin distributions after unreplicating v_1 from V_B

the cutsize is still 1. The gain values and the pin distributions after running Algorithm 5 are shown in Fig. 4.8.

4.1.3.5 Complexity Analysis of rFM

A single pass of rFM consists of initial gain computation, repeatedly selecting a vertex to operate on and gain updates of neighbors of the selected vertex after performing an operation on that vertex (see Algorithm 1). The total number of vertices is equal to the sum of the number of non-replicated vertices and the number of replicated vertices. This is because, in our implementation, when a vertex is replicated, the new replica of this vertex is not added to the data structure of the current hypergraph throughout the corresponding uncoarsening phase. For a replicated vertex connected to a net, there is only a single pin of that net for the replicated vertex. Thus, clearly, the number of vertices and the number of pins of the hypergraph at the beginning of a coarsening phase will be equal to the number of vertices and the number of pins of the hypergraph at the end of

the corresponding uncoarsening phase. Section 4.3 explains how the replicated vertices and their pins are handled in the construction of sub-hypergraphs for further bisections.

Let n and p be the number of vertices and the number of pins of a given bipartition $\Pi^R = \{V_A, V_B\}$ on $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ for a pass of rFM. Let r be the number of replicated vertices and s be the number of non-replicated vertices. Clearly, $n = r + s$. The initial gain computation takes $O(p)$ time since in Algorithm 2, the pins of each net are traversed. After the initial gain computation is completed, these gain values are stored in 6 heaps. For each heap, it is required to perform build-heap operations. The build-heap operations on heapMA and heapMB will take a total of $O(s)$ time and similarly, the build-heap operations on heapRA and heapRB will take a total of $O(s)$ time since the number of vertices on heapMA and heapMB or heapRA and heapRB is equal to number of non-replicated vertices, s (see Section 4.1.2 for the abbreviations of the heaps we use). The build-heap operation on heapUA will take $O(r)$ time and similarly, the build-heap operation on heapUB will take $O(r)$ time since each heap possesses r number of elements. Thus, the total time required for building heaps is equal to $O(r + r + s + s) = O(2n) = O(n)$.

The selection procedure consists of checking maximum gain values in 6 heaps, which takes constant amount of time. After selecting the gain value from one of the heaps with respect to the selection criteria, we perform an extract-max operation on the selected heap and a delete operation on another heap for the other gain value of the selected vertex (Section 4.1.2). No matter the selected heap, the extract-max and delete operations on the heaps will be bounded by the number of total vertices since the maximum number of elements in a single heap can be at most n. Thus, a single selection operation will take $O(1 + 2 \log n) = O(\log n)$. In a single pass of rFM where all vertices are exhausted, we can make at most n selections. Consequently, the cost of selection in a single pass of rFM is equal to $O(n \log n)$.

In the original move based FM algorithm [28], the gain updates take $O(p)$ time since in a single pass, there are at most three update operations for the vertices

connected to n_j and, the move gain of a vertex connected to n_j is updated at most twice on these three update operations. This is due to the critical net definitions for the move operations and the locking schemes the FM algorithm uses [28]. In rFM, we use the same critical net definitions and locking schemes as the original FM algorithm. The critical net definitions of the newly introduced replication and unreplication operations are subset of the critical net definitions of the move operation (see Section 4.1.3). Thus, the complexity of gain updates in a single pass of rFM is the same as the FM heuristic. In rFM, since each vertex possesses two gains, in the worst case, both of these gain values may need to be updated, which doubles the number of gain updates for any vertex compared to FM. Each gain update requires an increase-key or decrease-key operations on the corresponding heap. Consequently, the complexity of rFM is $O(2p \log n) = O(p \log n)$. This is equal to the complexity of the heap implementation of original FM where gains are stored in heaps instead of buckets.

Given these complexity values, the complexity of a single pass of rFM is $O(n + n \log n + p \log n) = O(p \log n)$ since $p \geq n$.

In our implementation, the space that rFM requires is modest. For each vertex, an additional gain value is stored compared to original FM. For each net, it is necessary to store an extra field that indicates the number of replicated vertices in addition to the number of non-replicated vertices that are in part A and B.

4.2 rFM and Multilevel Framework

The multilevel framework for HP consists of 3 phases as mentioned in Chapter 2.3. The replication is achieved in the uncoarsening phase of the multilevel scheme where the refinement algorithm rFM is used as a replication tool. The coarsening and the initial partitioning phases are used as-is since they do not include the replication process.

At each level of the uncoarsening phase, we perform multiple passes to refine

the current bipartition. At the end of each level, the bipartition Π_i^R on the coarser hypergraph $\mathcal{H}_i$ is projected back to the bipartition Π_{i-1}^R on the finer hypergraph $\mathcal{H}_{i-1}$. The projection includes the decomposition of each super-vertex in $\mathcal{H}_i$ to its constituent vertices in $\mathcal{H}_{i-1}$. The decomposition of a non-replicated super-vertex in $\mathcal{H}_i$ results in multiple non-replicated vertices in $\mathcal{H}_{i-1}$. Similarly, the decomposition of a replicated super-vertex in $\mathcal{H}_i$ results in multiple replicated vertices in $\mathcal{H}_{i-1}$. The existence of replicated vertices does not disturb the projection process. Clearly, the decomposition of a replicated super-vertex to its constituent replicated vertices will not change the cut-state of the nets this replicated super-vertex is connected to. Furthermore, the single pin nets which are eliminated in the coarsening phase of the corresponding level will occur in the finer hypergraph. If this single pin is a replicated vertex, such nets will have only replicated vertices connected to it in the finer hypergraph and thus, the single pin nets will still be internal.

Unnecessary replicas tend to occur excessively at the beginning of each uncoarsening level due to the increase in the degrees of freedom after the projection of a coarser hypergraph to a finer hypergraph. Such replicas hamper the refinement and partitioning if they are not removed, since:

- They consume the given replication amount needlessly which may prevent the positive gain replications to be performed.
- In the construction of the new hypergraphs for further bipartitions, they can cause the new hypergraphs to become unnecessarily bigger.

In the operation selection, we give the unreplication of unnecessary replicas the highest priority (Section 4.1.2). By giving this operation the highest priority, the majority of the unnecessary replicas are eliminated at the beginning of each uncoarsening level.

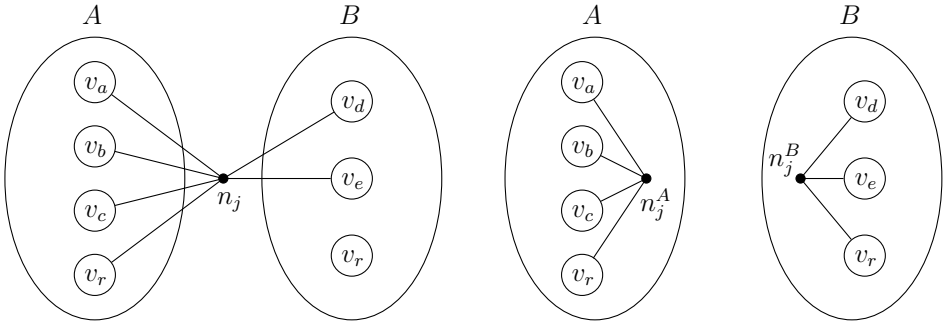
4.3 Recursive Bipartitioning and Replica Selection

The K-way HP problem is generally solved with Recursive Bipartitioning framework (Chapter 2.3). In the RB scheme, firstly a 2-way partition of the initial hypergraph is obtained and then, the obtained parts are bipartitioned in a recursive manner until reaching K parts. We use the same scheme to obtain multi-way partitioning of a given hypergraph for the replicated HP problem. After each bipartition, two new hypergraphs are constructed from the parts of the bipartition for further bipartitioning. In the construction of the new hypergraphs, the replicas of the replicated vertices become non-replicated vertices for the new hypergraphs and the necessary pins are placed for these vertices. Consequently, the weight and the number of pins of the resulting hypergraphs are greater when compared to the HP without replication.

4.3.1 Cut-net Splitting

In the RB framework, the cut-net splitting scheme is used in order to capture the connectivity cutsize metric. After each bipartitioning, two sub-hypergraphs, $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$, are constructed from $\Pi^R = \{V_A, V_B\}$. The vertex sets of $\mathcal{H}_\mathcal{A}$ and $\mathcal{H}_\mathcal{B}$ are equivalent to V_A and V_B respectively. Clearly, the internal nets in A will be in the net set of $\mathcal{H}_\mathcal{A}$ and the internal nets in B will be in the net set of $\mathcal{H}_\mathcal{B}$. Each cut-net net $n_j \in \Pi^R$ is split into two nets, n_j^A and n_j^B, where $Pins(n_j^A) = Pins(n_j) \cap V_A$ and $Pins(n_j^B) = Pins(n_j) \cap V_B$. Then, n_j^A is added to the net list of $\mathcal{H}_\mathcal{A}$ if $\sigma_A(n_j^A) > 1$ and n_j^B is added to the net list of $\mathcal{H}_\mathcal{B}$ if $\sigma_B(n_j^B) > 1$. Clearly, $\sigma_B(n_j^A) = 0$ and $\sigma_A(n_j^B) = 0$. Single pin nets are eliminated in splitting since they cannot contribute to cutsize in further bipartitions.

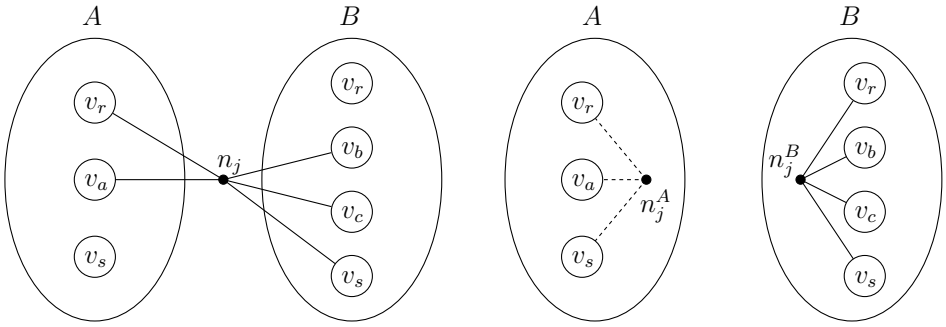
The cut-net splitting scheme is extended to include pins to the replicas of the replicated vertices. In the cut-net splitting scheme, where there are replicated vertices, we need to add pins to the replicas of the replicated vertices in order to preserve the flexibility of performing move or replication operations on them in



(a) Before net splitting. (b) After net splitting

Figure 4.9: Cut-net splitting, no pins of net n_j are discarded.

the newly constructed hypergraphs. After the split of a cut-net n_j, a pin is added for each replica of the replicated vertex connected by n_j in part A if $\sigma_A(n_j^A) > 1$ and for each replica of n_j in part B if $\sigma_B(n_j^B) > 1$. We do not add pin(s) to the replica(s) in part A if $\sigma_A(n_j^A) < 2$ since n_j^A cannot be cut in further bipartitions in $\mathcal{H}_\mathcal{A}$. Similarly, we do not add pin(s) to the replica(s) in part B if $\sigma_B(n_j^B) < 2$ since n_j^B cannot be cut in further bipartitions in $\mathcal{H}_\mathcal{B}$. Figs. 4.9 and 4.10 show the splitting two different cut-nets. In Fig. 4.9, the vertex v_r is replicated and the pins for the replicas of v_r need to be added for the split nets since $\sigma_A(n_j^A) = 3$ and $\sigma_B(n_j^B) = 2$. In Fig. 4.10, there are two replicated vertices, v_r and v_s. After the splitting of n_j, we do not need to add the pins to the replicas of v_r and v_s in part A since $\sigma_A(n_j^A) = 1$. However, we need to add pins to the replicas of v_r and v_s in part B since $\sigma_B(n_j^B) = 2$.



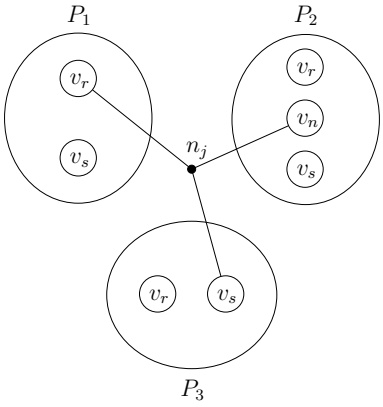
(a) Before net splitting. (b) After net splitting

Figure 4.10: Cut-net splitting, the pins of net n_j in part A are discarded.

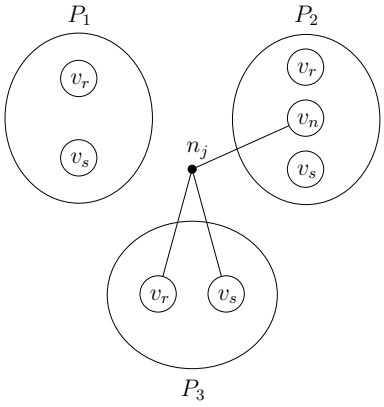
4.3.2 Replica Selection

The replication of a vertex v_i brings the problem of selecting replicas of v_i for each net it is connected by. If a net n_j connects replicated vertices, we need to decide which replicas of these replicated vertices will be used by n_j. This is required for a couple of reasons: (i) the cut computation of the final partition and; (ii) the investigated real world problem may enforce the nets to make a choice from which parts their replicas will be used. We propose a simple replica selection technique. The basic motivation behind this technique is not to increase the cutsize with careless replica selection. Fig. 4.11 shows various replica selection alternatives for n_j in a 3-way partition. There are two replicated vertices connected to n_j, v_r and v_s, each having three replicas and, there is a non-replicated vertex connected to n_j in part P_2, v_n. Clearly, λ_j will be at least one because of the non-replicated vertex in P_2. If we select the replica of v_r in P_1 and the replica of v_s in P_3 for n_j as in Fig. 4.11a, n_j is cut ($\lambda_j = 3$) and the contribution of n_j to the cutsize is $2c(n_j)$. Another selection alternative may be to select the both replicas of v_r and v_s in P_3 as in Fig. 4.11b. In this case, n_j is cut ($\lambda_j = 2$) and the contribution of n_j to the cutsize is $c(n_j)$. The logical selection alternative is seen in Fig. 4.11c, where the both replicas of v_r and v_s are selected from P_2, from the part of v_n. In this replica selection alternative, n_j is uncut ($\lambda_j = 1$) and n_j does not contribute to the cutsize of the partition. This example shows how replica selection can be crucial in computing cutsize of given partitions.

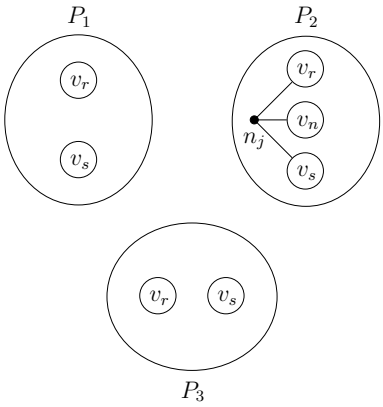
Our replica selection technique consists of evaluating each net's replicas and making a decision about which one to use after obtaining a K-way partitioning. The replica selection decision is based on the pin distributions of nets. Consider a net n_j and a replicated vertex v_i connected by it that has n replicas, $r_1, \ldots, r_n$, we are to pick one of n replicas of v_i for n_j. For each r_i, we count the number of non-replicated and replicated vertices connected by n_j which are in the same part with r_i. Then, we pick the replica with the highest number of non-replicated vertices. Selecting such a replica will not increase the cutsize since the part of the selected replica already contributes to cutsize for n_j because of the non-replicated vertices connected to n_j in that part. If the number of non-replicated vertices



(a) Careless replica selection for n_j ($\lambda_j = 3$).



(b) Another careless replica selection for n_j ($\lambda_j = 2$).



(c) Careful replica selection for n_j ($\lambda_j = 1$).

Figure 4.11: Three replica selection alternatives for n_j.

connected to n_j which are in the same part with r_i is zero for all replicas, we pick the replica with the highest number of replicated vertices which are connected to n_j and in the same part with r_i. In this way, we assure n_j to select its replicas from the part which possesses the highest number of replicated vertices connected to n_j. If this value is zero too, then we randomly pick one of the replicas of v_i for n_j. Fig. 4.12 shows an example of this selection technique on nets n_j and n_k in a 4-way partition. There are two replicated vertices, v_r, v_s and, three non-replicated vertices, v_m, v_n, v_p. After the replica selections are done for n_j and n_k with respect to the criteria mentioned above, $\lambda_j = 3$ and $\lambda_k = 2$.

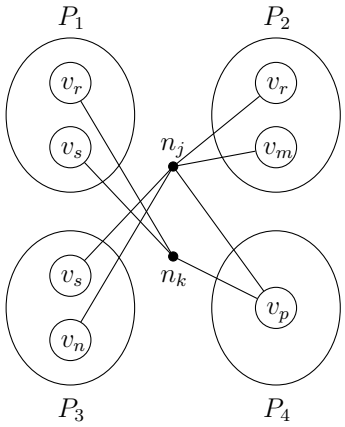


Figure 4.12: Replica selection for n_j ($\lambda_j = 3$) and n_k ($\lambda_k = 2$).

4.3.3 Replication Amount Distribution

The RB scheme consists of multiple bipartitions. The replication amount used in each bipartitioning can have an effect on the cutsize of the final partition. We try 2 different replication amount distribution schemes in this work:

- *Level-wise Replication.* The replication amount is distributed evenly among the levels of the RB. Firstly, the total replication amount is divided by the number of levels, $\lg_2 K$. Then, for each specific level, the replication amount is evenly distributed among the hypergraphs in this level.
- *Bisection-wise Replication.* In this scheme, each bipartitioning possesses the same amount of replication.

Chapter 5

Experimental Results

In this chapter, we conduct experiments to test the performance of the proposed replication scheme on the cutsize and balance of the partitions. Firstly, we briefly explain the integration of our replication scheme into the multilevel hypergraph partitioning tool PaToH [15] and give details of the experimental setup. Then, we discuss the properties of the datasets used in the experiments. In the experiments, we compare two important quality metrics, cutsize and imbalance, of the partitions with and without replication. We also evaluate the gradient methodology described in the previous chapter in our experiments.

5.1 Experimental Setup

The proposed replication scheme is implemented and integrated into the multilevel HP tool PaToH. This version of PaToH is capable of vertex replication and, we call it replicated PaToH (repl-PaToH). As mentioned in previous chapters, replication is achieved in the uncoarsening phase of the multilevel methodology. In the coarsening and the initial partitioning phases, PaToH is used as-is, however, the uncoarsening phase is written from scratch. In our experiments, we use the same parameters for PaToH and repl-PaToH in the coarsening and the initial

partitioning phases. We use agglomerative clustering (absorption clustering using pins) and greedy hypergraph growing partition algorithms in the coarsening and initial partitioning phases, respectively. The initial partitioning algorithm is run multiple times and the bipartition with minimum cutsize is selected for the uncoarsening phase.

The parameters that are used in the uncoarsening phase are worth to mention for both versions:

- *Refinement algorithm,* is the algorithm that is used as the refinement algorithm in the uncoarsening phase. For PaToH, boundary FM (BFM) and boundary KL (BKL) algorithm is used. For repl-PaToH, we use the rFM algorithm described in the previous chapter.

- *Initial and final imbalance,* are the values that must be assured at the beginning and at the end of each uncoarsening phase, respectively. Maintaining a loose initial imbalance value may help FM based heuristics to work better since at the beginning levels, the average weight of a single vertex is greater and performing an operation on a vertex may be prevented due to tight size constraints on the parts. Thus, the initial imbalance is set to 0.12 and the final imbalance is set to 0.10 for PaToH and repl-PaToH.

- *Number of passes,* is the value that how many times the refinement algorithm is run at each level of the uncoarsening phase. We set this parameter to 3 for PaToH and repl-PaToH in our experiments.

- *Window size,* is the number of operations that is allowed to be performed which do not improve the cutsize. The window size is set to 100 for both PaToH and repl-PaToH.

The experiments are conducted on a 4 $\times$ AMD Six-Core Opteron, each core having a clock frequency of 2.1 GHz, 64 KB L1 Instruction and Data cache, 512 KB L2 cache and 6 MB shared L3 cache. Each processor has a 32 GB of memory which makes a total of 128 GB memory. The implementation is done

Data set	# vertices	# nets	# pins	Avg. net deg.	Avg. net weight	Avg. vertex weight
California HPN	10141	17369	52965	3.05	13.36	48.76
Minnesota7	34222	46056	149493	3.25	26.86	47.11
New Mexico	448959	510477	1682429	3.30	17.32	43.64
Oldenburg	4465	7886	22085	2.80	15.06	42.62
Oregon	507212	574353	1844579	3.21	17.96	41.96
San Francisco	166558	197630	654990	3.31	18.34	44.99
San Joaquin	17444	26225	80785	3.08	16.66	46.17
Washington	548901	613598	1980535	3.23	21.77	42.01
Wyoming	254648	302494	984039	3.25	17.53	43.85
CalGovernerRecall	172556	30805	3545766	115.10	1.00	1.00
Facebook	4618974	66568	14277456	214.48	1.00	1.00
Stanford	281903	281903	2312497	8.20	1.00	8.20

Table 5.1: The properties of the data sets used in our experiments.

in C programming language and all files are compiled with gcc –O3 optimization flag enabled.

5.2 Datasets

In the experiments, we test our algorithm on various datasets from spatial network and information retrieval fields. Various characteristics of these datasets are depicted in Table 5.1 such as number of vertices, nets and pins, average net degree, average net and vertex weight. There are 9 data sets for spatial networks that include data sets from US Department of Transportation [23] (California HPN dataset), US Tiger/Line [16] (Minnesota7 that includes data from 7 countries, New Mexico, Oregon, San Francisco, Washington and Wyoming datasets) and Brinkhoff's network data generator [9] (Oldenburg and San Joaquin datasets). There are 3 data sets for information retrieval. Two of them are crawled using Stanford WebBase Project [17, 54] (Facebook and CalGovernerRecall datasets) and the other one is from University of Florida Sparse Matrix Collection [19] (Stanford dataset which depicts the Stanford web graph).

The basic difference between the spatial network and IR datasets is clearly

the average net degree where in IR datasets this value is much higher. In the hypergraph models for road networks, the junctions are considered as both nets and vertices and each junction has an average of about three roads connected to it when we consider the datasets in Table 5.1. However, in IR based hypergraph models, generally the nets represent the documents and the pins of a net represent the terms/links in that document. From this perspective, it is obvious that average degree of a net for the hypergraph models in IR will be higher than those in spatial networks.

5.3 Results

The results include two important quality metrics in HP: cutsize and imbalance. We illustrate the results for $K = 32, 64, 128, 256$ and $\rho = 0.05, 0.10, 0.20$. Firstly, we compare PaToH and two replication amount distribution schemes (bisection-wise and level-wise) mentioned in Chapter 4. Then, rFM heuristic and the gradient methodology is compared. Each instance is run 10 times and the average of these runs is shown in the results.

Tables 5.2, 5.3, 5.4 and 5.5 show the cutsize and imbalance values of PaToH and repl-PaToH with two different replication schemes, bisection-wise and level-wise replication. Figs. 5.1, 5.2, 5.3 and 5.4 illustrate the improvement in the cutsize of two different replication schemes which are represented in the mentioned tables. In spatial network datasets, the improvement in the cutsize is greater than the improvement in the IR datasets. This is mainly due to the difference of average net degrees between these datasets. In IR datasets, the average net degree is high and saving a net from cut is harder compared to the nets in spatial network datasets. For example, in Table 5.3, for spatial network datasets, the average improvement of repl-PaToH (bisection-wise) for $\rho = 0.05, 0.10$ and, 0.20 is $56.52, 62.55$ and, 62.32, respectively whereas for IR datasets, these values are $13.22, 17.87$ and, 22.62, respectively. The values in other tables indicate similar results where the improvement in the cutsize in IR datasets is lower. As ρ grows higher, generally, the cutsize decreases for a specific value of K since

Dataset	ρ	PaToH		repl-PaToH, bisection-wise			repl-PaToH, level-wise		
		Cut	Imb.	Cut	Cut Imp. (%)	Imb.	Cut	Cut Imp. (%)	Imb.
California HPN	0.05	13810	5.72	6501	52.92	4.78	8791	36.34	5.18
	0.10	13896	5.49	3599	74.10	5.29	6559	52.79	5.59
	0.20	13921	6.21	3368	75.80	7.26	4693	66.28	5.76
Minnesota7	0.05	20542	5.11	4252	79.30	5.98	6868	66.56	4.55
	0.10	20279	5.14	4969	75.49	4.04	6335	68.76	4.09
	0.20	20118	5.28	7000	65.20	5.08	7081	64.80	3.79
New Mexico	0.05	23241	6.72	8900	61.70	4.54	10434	55.10	3.24
	0.10	23210	6.16	10033	56.77	2.85	11800	49.15	1.80
	0.20	23163	6.18	11949	48.41	2.01	13633	41.14	1.15
Oldenburg	0.05	10348	6.34	5751	44.42	5.45	6701	35.24	5.37
	0.10	10325	5.98	3247	68.55	5.28	5054	51.05	4.91
	0.20	10406	6.05	2665	74.38	4.98	3658	64.84	5.10
Oregon	0.05	29125	6.30	11584	60.22	4.67	12790	56.08	3.62
	0.10	28550	6.02	13071	54.21	2.64	14305	49.89	1.86
	0.20	28704	6.98	13958	51.37	1.64	17286	39.77	1.24
San Francisco	0.05	25402	6.40	6827	73.12	4.13	7854	69.08	4.02
	0.10	25393	6.48	7938	68.73	2.92	8225	67.60	2.70
	0.20	25361	7.01	9124	64.02	2.64	10035	60.43	1.81
San Joaquin	0.05	17723	6.20	5898	66.72	6.02	8617	51.37	5.09
	0.10	17417	5.57	4141	76.22	6.29	6324	63.69	4.72
	0.20	17582	6.55	4631	73.66	5.54	5328	69.69	4.96
Washington	0.05	27088	6.48	8611	68.21	4.06	9448	65.12	3.05
	0.10	27114	5.99	10206	62.35	3.26	11859	56.26	1.63
	0.20	27251	6.33	12186	55.28	2.00	13076	52.01	1.83
Wyoming	0.05	28821	6.08	11998	58.37	3.73	14121	51.00	3.50
	0.10	28746	6.43	14461	49.69	2.70	15399	46.43	1.85
	0.20	28747	6.42	15907	44.66	1.51	17794	38.10	0.83
AVERAGE	0.05		6.15		62.78	4.82		53.99	4.18
	0.10		5.92		65.12	3.92		56.18	3.24
	0.20		6.33		61.42	3.63		55.23	2.94
CalGovernerRecall	0.05	90677	3.19	87866	3.10	9.10	90147	0.58	8.18
	0.10	92010	3.66	84604	8.04	9.89	87368	5.04	8.47
	0.20	90056	3.65	81315	9.70	9.56	85353	5.22	9.05
Facebook	0.05	173581	0.00	159872	7.89	9.06	168977	2.65	8.99
	0.10	174658	0.00	158727	9.12	8.59	160176	8.29	8.57
	0.20	176880	0.00	153042	13.47	7.51	154873	12.44	8.29
Stanford	0.05	5552	4.62	3779	31.93	9.36	4138	25.46	8.95
	0.10	5563	4.71	3302	40.64	9.82	3847	30.84	9.78
	0.20	5451	4.94	3012	44.74	9.29	3665	32.76	8.37
AVERAGE	0.05		2.60		14.31	9.17		9.56	8.71
	0.10		2.79		19.27	9.43		14.72	8.94
	0.20		2.86		22.64	8.79		16.81	8.57
AVERAGE (All)	**0.05**		**5.26**		**50.66**	**5.91**		**42.88**	**5.32**
	0.10		**5.14**		**53.66**	**5.30**		**45.82**	**4.66**
	0.20		**5.47**		**51.72**	**4.92**		**45.62**	**4.35**

Table 5.2: The cut and imbalance values for $K = 32$.

Dataset	ρ	PaToH		repl-PaToH, bisection-wise			repl-PaToH, level-wise		
		Cut	Imb.	Cut	Cut Imp. (%)	Imb.	Cut	Cut Imp. (%)	Imb.
California HPN	0.05	24882	5.07	16074	35.39	5.38	18888	24.08	5.35
	0.10	24847	5.04	10208	58.91	5.78	15319	38.34	5.31
	0.20	25121	5.57	6769	73.05	8.37	11181	55.49	6.14
Minnesota7	0.05	38226	5.09	10623	72.21	5.51	17793	53.45	5.40
	0.10	38176	5.60	7851	79.43	6.88	13493	64.65	4.93
	0.20	37664	6.34	9889	73.74	6.40	12669	66.36	5.67
New Mexico	0.05	40443	6.57	15469	61.75	4.08	17505	56.71	3.81
	0.10	40422	6.20	17420	56.90	2.96	19423	51.94	1.96
	0.20	40443	6.50	20233	49.97	2.35	22222	45.05	1.75
Oldenburg	0.05	16866	5.18	11929	29.27	6.51	13273	21.30	5.30
	0.10	16767	5.86	7546	54.99	5.34	10423	37.83	4.57
	0.20	16690	6.02	5016	69.94	6.86	7557	54.72	5.06
Oregon	0.05	47694	6.52	18868	60.43	3.70	20735	56.52	3.90
	0.10	47565	6.94	21179	55.47	3.15	22759	52.15	1.98
	0.20	47592	6.54	23926	49.72	2.48	27158	42.93	1.34
San Francisco	0.05	44665	6.39	11385	74.51	5.02	16438	63.19	4.56
	0.10	44815	6.58	13755	69.30	3.96	15903	64.51	3.08
	0.20	44603	6.33	15672	64.86	3.11	18197	59.20	2.92
San Joaquin	0.05	29823	6.00	14955	49.85	6.17	19189	35.65	5.63
	0.10	29838	5.74	8278	72.25	6.92	14543	51.26	5.09
	0.20	29939	5.61	7643	74.47	8.90	10830	63.82	5.66
Washington	0.05	47565	6.09	16177	65.98	4.52	17683	62.82	3.82
	0.10	47027	5.76	17646	62.47	3.89	19986	57.50	2.17
	0.20	47639	5.95	20502	56.96	2.64	23064	51.58	1.65
Wyoming	0.05	47327	6.12	19270	59.28	4.53	21753	54.03	3.85
	0.10	47439	6.33	22196	53.21	3.11	24557	48.23	2.24
	0.20	47671	5.67	24706	48.17	2.45	28956	39.25	1.50
AVERAGE	0.05		5.89		56.52	5.05		47.53	4.63
	0.10		6.01		62.55	4.67		51.82	3.48
	0.20		6.06		62.32	4.84		53.16	3.52
CalGovernerRecall	0.05	139097	4.21	135886	2.30	9.14	136342	1.98	8.58
	0.10	138494	4.53	132041	4.65	9.39	135345	2.27	8.91
	0.20	138690	4.71	127518	8.05	9.92	132349	4.57	8.07
Facebook	0.05	225148	0.73	205894	8.55	9.88	216265	3.94	9.08
	0.10	226226	0.47	197657	12.62	9.70	216342	4.36	8.64
	0.20	225076	0.66	190866	15.19	9.15	197758	12.13	8.01
Stanford	0.05	9546	4.33	6794	28.82	8.74	7503	21.40	8.06
	0.10	9485	4.39	6038	36.34	9.08	6777	28.55	8.45
	0.20	9475	4.51	5248	44.61	9.44	6459	31.83	8.78
AVERAGE	0.05		3.09		13.22	9.25		9.11	8.57
	0.10		3.13		17.87	9.39		11.77	8.67
	0.20		3.29		22.62	9.50		16.18	8.29
AVERAGE (All)	**0.05**		**5.19**		**45.70**	**6.10**		**37.92**	**5.61**
	0.10		**5.29**		**51.38**	**5.85**		**41.80**	**4.78**
	0.20		**5.37**		**52.39**	**6.01**		**43.91**	**4.71**

Table 5.3: The cut and imbalance values for $K = 64$.

Dataset	ρ	PaToH		repl-PaToH, bisection-wise			repl-PaToH, level-wise		
		Cut	Imb.	Cut	Cut Imp. (%)	Imb.	Cut	Cut Imp. (%)	Imb.
California HPN	0.05	41424	5.95	31248	24.56	6.13	35072	15.33	5.55
	0.10	41307	6.41	23458	43.21	5.61	30324	26.58	5.26
	0.20	41464	5.86	14322	65.45	7.40	23808	42.58	5.89
Minnesota7	0.05	67591	5.93	28680	57.56	5.52	42993	36.39	5.44
	0.10	67641	5.61	18017	73.36	7.19	33097	51.06	5.11
	0.20	67517	5.77	14903	77.92	8.42	25824	61.75	5.96
New Mexico	0.05	68009	5.95	24827	63.49	4.69	27926	58.93	4.16
	0.10	67586	6.25	27821	58.83	3.94	30373	55.06	2.63
	0.20	67860	6.53	31490	53.59	2.92	35969	46.99	2.09
Oldenburg	0.05	27044	5.62	22345	17.37	6.21	23814	11.94	5.38
	0.10	27016	5.97	18224	32.54	6.00	20995	22.28	5.28
	0.20	26909	6.13	11130	58.63	7.01	16478	38.76	5.27
Oregon	0.05	76467	6.28	28145	63.19	4.72	33560	56.11	4.12
	0.10	76594	5.84	32118	58.06	4.68	35731	53.35	2.94
	0.20	76057	6.37	35735	53.01	3.79	39863	47.58	2.49
San Francisco	0.05	76022	6.43	20551	72.96	5.45	35533	53.25	4.43
	0.10	76634	6.71	22367	70.81	5.91	29326	61.73	4.02
	0.20	76264	6.22	26000	65.90	5.22	28869	62.14	4.48
San Joaquin	0.05	49140	5.46	31822	35.24	6.11	37108	24.48	5.40
	0.10	49292	5.99	20612	58.18	7.29	29915	39.31	5.12
	0.20	49106	5.71	13678	72.14	9.34	22753	53.66	5.71
Washington	0.05	84776	5.49	26875	68.29	5.05	31838	62.44	4.02
	0.10	85026	6.27	29485	65.32	4.09	33335	60.79	2.83
	0.20	84980	6.28	33831	60.18	3.13	37748	55.58	2.17
Wyoming	0.05	75084	5.68	27924	62.80	4.75	35813	52.30	4.41
	0.10	75483	5.92	31904	57.73	4.05	37585	50.20	2.92
	0.20	75303	5.57	36891	51.00	3.60	42084	44.11	2.45
AVERAGE	0.05		5.87		51.72	5.40		41.24	4.77
	0.10		6.11		57.56	5.42		46.71	4.01
	0.20		6.05		61.98	5.65		50.35	4.06
CalGovernerRecall	0.05	208593	5.00	202054	3.13	9.22	209112	-0.24	8.73
	0.10	210678	4.89	199267	5.41	9.98	204978	2.70	8.26
	0.20	208122	5.04	198644	4.55	9.38	206481	0.78	8.67
Facebook	0.05	290153	1.69	259257	10.64	9.57	262754	9.44	9.19
	0.10	286970	1.30	256722	10.54	9.19	261865	8.74	8.42
	0.20	285888	1.37	251918	11.88	9.58	269446	5.75	9.36
Stanford	0.05	14739	4.56	11479	22.11	8.91	12517	15.07	8.00
	0.10	14813	4.72	10354	30.10	9.22	11708	20.96	9.09
	0.20	14804	4.49	9202	37.84	9.99	10818	26.92	9.54
AVERAGE	0.05		3.75		11.96	9.23		8.09	8.64
	0.10		3.64		15.35	9.46		10.80	8.59
	0.20		3.64		18.09	9.65		11.15	9.19
AVERAGE (All)	**0.05**		**5.34**		**41.78**	**6.36**		**32.95**	**5.74**
	0.10		**5.49**		**47.01**	**6.43**		**37.73**	**5.16**
	0.20		**5.45**		**51.01**	**6.65**		**40.55**	**5.34**

Table 5.4: The cut and imbalance values for $K = 128$.

Dataset	ρ	PaToH		repl-PaToH, bisection-wise			repl-PaToH, level-wise		
		Cut	Imb.	Cut	Cut Imp. (%)	Imb.	Cut	Cut Imp. (%)	Imb.
California HPN	0.05	66204	5.59	57949	12.46	5.93	60868	8.05	5.54
	0.10	66342	5.67	47508	28.38	6.79	55299	16.64	5.26
	0.20	66326	5.67	33506	49.48	7.53	46024	30.60	6.09
Minnesota7	0.05	115126	5.73	68613	40.40	5.95	86645	24.73	5.15
	0.10	115618	5.30	43859	62.06	6.93	71559	38.10	5.76
	0.20	115198	5.58	29059	74.77	9.21	55411	51.89	6.70
New Mexico	0.05	115295	5.72	35863	68.89	6.07	52043	54.86	4.66
	0.10	114937	5.74	41882	63.56	5.19	48228	58.03	3.41
	0.20	115017	6.10	47545	58.66	4.40	54443	52.66	2.94
Oldenburg	0.05	44058	6.32	42680	3.12	7.37	41283	6.29	5.53
	0.10	44113	6.32	39742	9.90	6.58	38277	13.22	5.51
	0.20	44063	6.05	35483	19.47	7.81	33825	23.23	6.25
Oregon	0.05	123780	6.18	40339	67.41	6.04	55504	55.15	4.36
	0.10	123141	6.23	47350	61.54	5.14	55977	54.54	3.59
	0.20	123376	5.91	52547	57.40	4.84	59520	51.75	4.19
San Francisco	0.05	125656	5.73	42435	66.22	5.95	71058	43.45	4.63
	0.10	125447	5.98	34045	72.86	7.13	55649	55.63	4.59
	0.20	125553	6.83	38604	69.25	7.17	47008	62.55	5.68
San Joaquin	0.05	79818	5.77	61244	23.27	6.24	68954	13.61	5.46
	0.10	79766	5.50	46238	42.03	6.21	58662	26.45	4.84
	0.20	80077	5.87	28480	64.43	8.58	46752	41.61	5.46
Washington	0.05	144799	5.97	40308	72.16	5.32	58844	59.36	4.17
	0.10	144929	6.17	46788	67.71	5.61	54376	62.48	3.50
	0.20	145185	5.93	53653	63.04	4.91	59881	58.75	3.77
Wyoming	0.05	120624	5.48	40955	66.04	5.62	66101	45.20	4.41
	0.10	120407	5.74	45552	62.16	5.44	57317	52.39	3.69
	0.20	120791	5.92	52501	56.53	4.93	60785	49.67	3.70
AVERAGE	0.05		5.83		46.66	6.06		34.52	8.78
	0.10		5.85		52.24	6.11		41.94	9.47
	0.20		5.99		57.00	6.60		46.97	10.14
CalGovernerRecall	0.05	305429	5.76	301908	1.15	9.87	309517	-1.33	8.89
	0.10	304936	6.10	300748	1.37	9.93	315186	-3.36	9.77
	0.20	304065	6.03	298685	1.76	10.67	308817	-1.56	10.06
Facebook	0.05	367075	2.79	343711	6.36	10.80	345023	6.00	9.59
	0.10	370908	2.69	337817	8.92	10.15	349106	5.87	9.63
	0.20	372214	2.53	340130	8.61	10.81	337885	9.22	9.93
Stanford	0.05	22233	4.97	18883	15.06	8.15	20072	9.71	7.87
	0.10	22121	4.08	17022	23.05	10.42	18831	14.87	9.01
	0.20	22200	4.41	15462	30.35	10.34	17867	19.51	10.42
AVERAGE	0.05		4.51		7.52	9.61		4.79	8.78
	0.10		4.29		11.11	10.17		5.79	9.47
	0.20		4.32		13.57	10.61		9.06	10.14
AVERAGE (All)	**0.05**		**5.50**		**36.88**	**6.94**		**27.09**	**5.86**
	0.10		**5.46**		**41.96**	**7.13**		**32.91**	**5.71**
	0.20		**5.57**		**46.15**	**7.60**		**37.49**	**6.27**

Table 5.5: The cut and imbalance values for $K = 256$.

with more replication, more nets can be saved from the cut. However, in some datasets, the opposite of this can happen where as ρ increases, the cutsize can decrease. For example in Table 5.4 for Oregon dataset with level-wise replication, the improvement in the cutsize for $\rho = 0.05, 0.10$ and, 0.20 is $56.11, 53.35$ and, 47.58, respectively. The main reason behind this anomaly is that if more replication amount is given to a hypergraph than the amount it needs, extra replication can lead to partitions with worse cutsize. In other words, $\rho = 0.10$ or 0.20 for the Oregon dataset gives more replication amount than this dataset needs and thus, the partitions have slightly worse cutsize values. For a specific ρ, as the value of K increases, the improvement in the cutsize decreases. For instance, for $\rho = 0.10$, the average cutsize values of all datasets for $K = 32, 64, 128$ and, 256 are $53.66, 51.38, 47.01$ and, 41.96, respectively. This is because as the number of parts increases, the possibility of a cut-net connecting more parts also increases. The imbalance values for repl-PaToH is close to PaToH for spatial network datasets while repl-PaToH has clearly higher imbalance values for IR datasets since in most of the cases, the given replication amount is not used uniformly in each bipartitioning for these datasets.

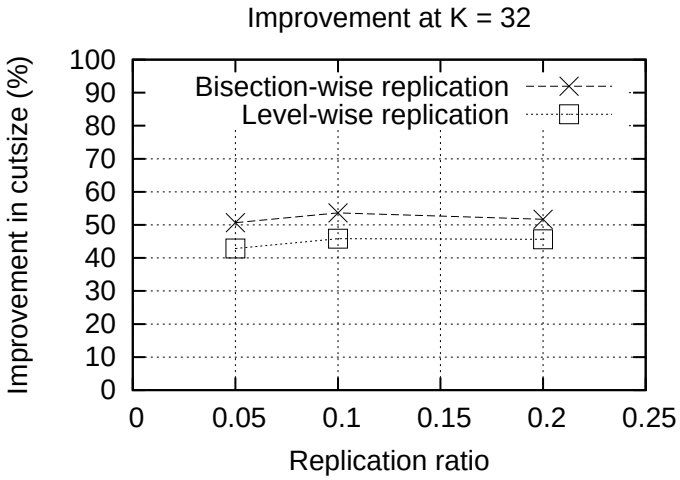


Figure 5.1: Improvement at $K = 32$.

Almost in all datasets, bisection-wise replication comes up with better cutsize values than the level-wise replication. In bisection-wise replication, the initial bipartitions have lower replication amount with respect to their total vertex weight

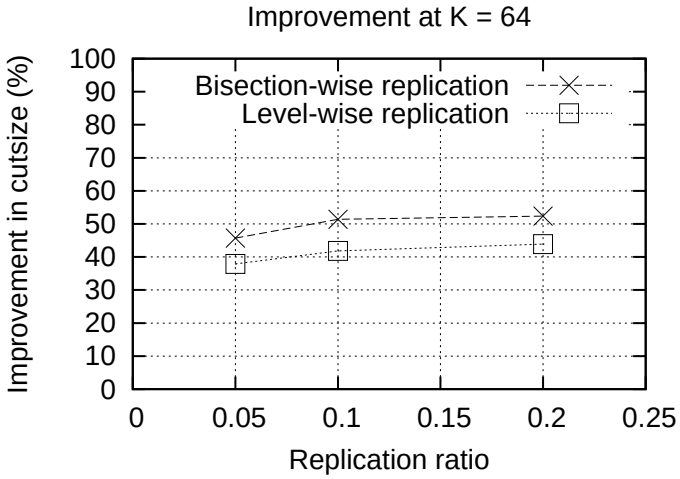


Figure 5.2: Improvement at $K = 64$.

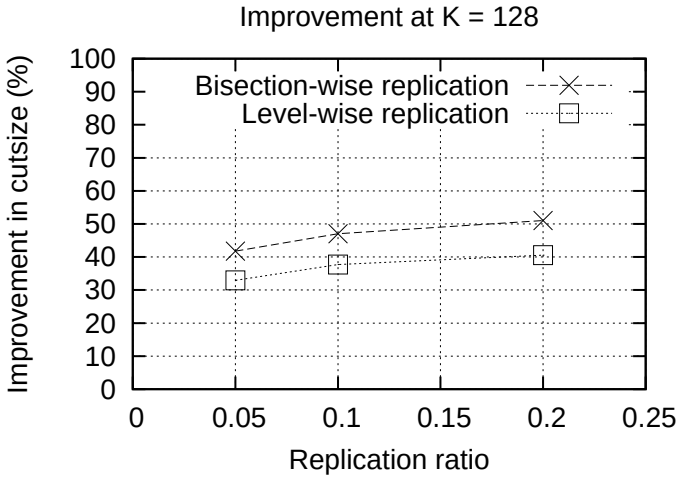


Figure 5.3: Improvement at $K = 128$.

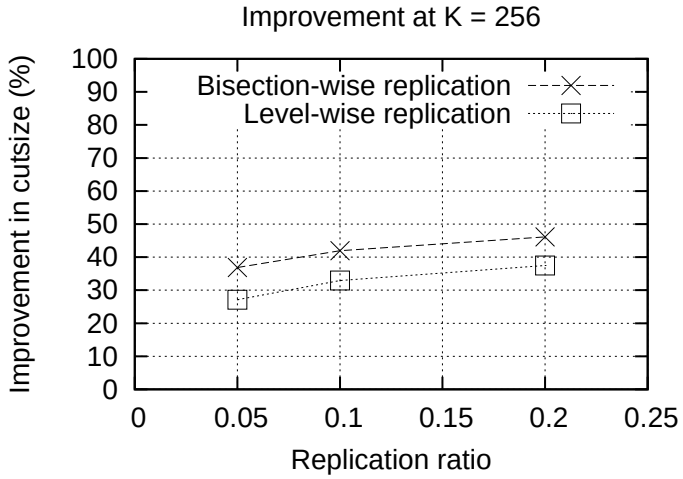


Figure 5.4: Improvement at $K = 256$.

whereas in level-wise replication, all bipartitions have equal replication amount. From this, we can conclude that performing more replication at deeper levels of the recursion tree can be more helpful in obtaining partitions with better cutsize. On the contrary, level-wise replication has generally better imbalance values. The reason behind this is that each bipartition gets equal replication amount with respect to their total vertex weight and after performing replication, the total vertex weights of the hypergraphs with replicated vertices are close to each other.

Tables 5.6, 5.7, 5.8 and 5.9 show the cutsize values of rFM and rFM with gradient methodology (gradient-rFM). We used the same datasets and tested them for both bisection-wise and level-wise replication. In the tables, we compare the replication schemes separately for rFM and gradient-rFM. The lower cutsize value for a specific replication scheme is illustrated as bold. For instance, in Table 5.7 for the Washington dataset with $\rho = 0.05$, gradient-rFM (15803) has a lower cutsize than rFM (16177) for bisection-wise replication whereas rFM (17683) has lower cutsize value than gradient-rFM (18438) for level-wise replication. Generally, rFM performs better than gradient-rFM in spatial network datasets. However, in IR datasets, gradient-rFM performs better; especially in CalGovernerRecall dataset where for almost all K values, gradient-rFM is superior to rFM. This can indicate that gradient-rFM may be well-suited to the hypergraphs with relatively higher average net degree.

Dataset	ρ	repl-PaToH, rFM		repl-PaToH, rFM-gradient	
		Cut (bisection-wise)	Cut (level-wise)	Cut (bisection-wise)	Cut (level-wise)
California HPN	0.05	**6501**	8791	6670	**8755**
	0.10	**3599**	**6559**	3829	6650
	0.20	**3368**	**4693**	3431	4715
Minnesota7	0.05	**4252**	**6868**	4826	7617
	0.10	**4969**	**6335**	5295	6621
	0.20	7000	**7081**	**6822**	7366
New Mexico	0.05	**8900**	**10434**	9077	10494
	0.10	**10033**	11800	10515	**11385**
	0.20	11949	**13633**	**11321**	14018
Oldenburg	0.05	5751	**6701**	**5653**	6759
	0.10	3247	5054	**3070**	**4989**
	0.20	**2665**	**3658**	2913	3662
Oregon	0.05	**11584**	**12790**	11767	12862
	0.10	13071	14305	**12807**	**14000**
	0.20	**13958**	17286	15068	**15776**
San Francisco	0.05	6827	**7854**	**6771**	7997
	0.10	7938	**8225**	**7780**	9241
	0.20	**9124**	**10035**	9240	10574
San Joaquin	0.05	5898	**8617**	**5740**	8741
	0.10	**4141**	**6324**	4373	6550
	0.20	**4631**	**5328**	4775	5529
Washington	0.05	**8611**	9448	9007	**8899**
	0.10	10206	11859	**9955**	**11221**
	0.20	**12186**	13076	12232	**12498**
Wyoming	0.05	**11998**	**14121**	12247	14810
	0.10	**14461**	**15399**	14788	15925
	0.20	**15907**	17794	16074	**17706**
CalGovernerRecall	0.05	87866	90147	**86487**	**88574**
	0.10	**84604**	87368	85192	**85050**
	0.20	81315	85353	**80648**	**83147**
Facebook	0.05	**159872**	168977	162041	**162831**
	0.10	**158727**	**160176**	158801	164579
	0.20	153042	**154873**	**151484**	156669
Stanford	0.05	3779	4138	**3725**	**4082**
	0.10	3302	3847	**3293**	**3798**
	0.20	**3012**	3665	3024	**3560**

Table 5.6: The cut values for rFM and gradient rFM for $K = 32$.

Dataset	ρ	repl-PaToH, rFM		repl-PaToH, rFM-gradient	
		Cut (bisection-wise)	Cut (level-wise)	Cut (bisection-wise)	Cut (level-wise)
California HPN	0.05	16074	18888	**15837**	**18830**
	0.10	10208	**15319**	**10133**	15445
	0.20	6769	11181	**6613**	**10976**
Minnesota7	0.05	10623	**17793**	**10325**	18229
	0.10	**7851**	13493	8238	**13335**
	0.20	9889	12669	**9800**	**12216**
New Mexico	0.05	15469	17505	**14915**	**16546**
	0.10	**17420**	19423	17522	**19297**
	0.20	20233	**22222**	**20084**	22451
Oldenburg	0.05	**11929**	**13273**	12119	13393
	0.10	**7546**	**10423**	7909	10668
	0.20	5016	**7557**	**4874**	7729
Oregon	0.05	**18868**	**20735**	19041	20798
	0.10	**21179**	**22759**	21217	22928
	0.20	**23926**	27158	24453	**26140**
San Francisco	0.05	**11385**	**16438**	12078	16455
	0.10	**13755**	15903	14222	**15785**
	0.20	**15672**	**18197**	16127	18254
San Joaquin	0.05	14955	19189	**14666**	**19141**
	0.10	**8278**	14543	9222	**14453**
	0.20	7643	**10830**	**7145**	11063
Washington	0.05	16177	**17683**	**15803**	18438
	0.10	17646	**19986**	**17249**	20342
	0.20	**20502**	23064	20812	**22730**
Wyoming	0.05	19270	**21753**	**18341**	23142
	0.10	22196	**24557**	**21730**	25165
	0.20	**24706**	28956	25160	**28049**
CalGovernerRecall	0.05	135886	136342	**131748**	**132269**
	0.10	132041	135345	**129128**	**132609**
	0.20	127518	132349	**123750**	**129782**
Facebook	0.05	205894	216265	**203023**	**206131**
	0.10	**197657**	**216342**	206643	222184
	0.20	**190866**	**197758**	206423	200718
Stanford	0.05	**6794**	7503	6871	**7413**
	0.10	6038	**6777**	**5993**	6878
	0.20	**5248**	6459	5382	**6368**

Table 5.7: The cut values for rFM and gradient rFM for $K = 64$.

Dataset	ρ	repl-PaToH, rFM		repl-PaToH, rFM-gradient	
		Cut (bisection-wise)	Cut (level-wise)	Cut (bisection-wise)	Cut (level-wise)
California HPN	0.05	**31248**	35072	31451	**34952**
	0.10	**23458**	**30324**	23724	30395
	0.20	**14322**	**23808**	14843	24027
Minnesota7	0.05	**28680**	**42993**	29124	43067
	0.10	**18017**	33097	18444	**33053**
	0.20	**14903**	**25824**	15998	26005
New Mexico	0.05	24827	**27926**	**23936**	27965
	0.10	**27821**	**30373**	28212	30668
	0.20	**31490**	**35969**	32399	36706
Oldenburg	0.05	**22345**	**23814**	22485	24043
	0.10	**18224**	**20995**	18353	21022
	0.20	11130	**16478**	**11020**	16527
Oregon	0.05	**28145**	33560	28586	**32318**
	0.10	**32118**	**35731**	32372	36021
	0.20	**35735**	**39863**	37040	40816
San Francisco	0.05	**20551**	35533	20914	**35023**
	0.10	22367	29326	**21957**	**28579**
	0.20	26000	**28869**	**25530**	28878
San Joaquin	0.05	**31822**	37108	32140	**37012**
	0.10	**20612**	**29915**	21095	30145
	0.20	**13678**	**22753**	13679	22810
Washington	0.05	26875	31838	**25822**	**31278**
	0.10	29485	**33335**	**29110**	34627
	0.20	33831	**37748**	**33222**	38087
Wyoming	0.05	**27924**	**35813**	27965	36304
	0.10	**31904**	37585	32810	**37234**
	0.20	36891	**42084**	**36692**	42609
CalGovernerRecall	0.05	202054	209112	**201544**	**205582**
	0.10	199267	204978	**196718**	**200668**
	0.20	198644	206481	**197480**	**201590**
Facebook	0.05	**259257**	262754	259425	**260767**
	0.10	**256722**	**261865**	267026	272611
	0.20	**251918**	**269446**	253770	272439
Stanford	0.05	11479	12517	**11402**	**12513**
	0.10	**10354**	11708	10453	**11547**
	0.20	9202	10818	**9084**	**10667**

Table 5.8: The cut values for rFM and gradient rFM for $K = 128$.

Dataset	ρ	repl-PaToH, rFM		repl-PaToH, rFM-gradient	
		Cut (bisection-wise)	Cut (level-wise)	Cut (bisection-wise)	Cut (level-wise)
California HPN	0.05	**57949**	60868	57954	**60765**
	0.10	**47508**	55299	48170	**55296**
	0.20	**33506**	**46024**	33847	46598
Minnesota7	0.05	**68613**	**86645**	69524	86676
	0.10	**43859**	71559	44825	**71486**
	0.20	29059	**55411**	**29016**	55946
New Mexico	0.05	**35863**	52043	35977	**51277**
	0.10	**41882**	**48228**	42475	49098
	0.20	**47545**	54443	47889	**53163**
Oldenburg	0.05	**42680**	**41283**	42913	41344
	0.10	39742	38277	**39734**	**38204**
	0.20	**35483**	33825	35644	**33743**
Oregon	0.05	**40339**	**55504**	40763	55618
	0.10	47350	55977	**47160**	**54991**
	0.20	52547	59520	**53203**	**59394**
San Francisco	0.05	**42435**	**71058**	42755	72628
	0.10	**34045**	**55649**	34896	56102
	0.20	**38604**	**47008**	39253	47328
San Joaquin	0.05	**61244**	**68954**	61788	69460
	0.10	**46238**	58662	46671	**58638**
	0.20	**28480**	**46752**	28842	47403
Washington	0.05	**40308**	**58844**	41157	60109
	0.10	**46788**	**54376**	47081	55802
	0.20	53653	59881	**52494**	**59022**
Wyoming	0.05	**40955**	66101	41226	**65738**
	0.10	**45552**	57317	46438	**57020**
	0.20	52501	**60785**	**52414**	61383
CalGovernerRecall	0.05	301908	309517	**293178**	**299822**
	0.10	300748	315186	**293819**	**301121**
	0.20	298685	308817	**287586**	**298014**
Facebook	0.05	**343711**	345023	355325	**340151**
	0.10	**337817**	349106	338080	**340675**
	0.20	340130	**337885**	**327400**	340727
Stanford	0.05	18883	**20072**	**18869**	20146
	0.10	17022	**18831**	**16952**	18835
	0.20	15462	17867	**15452**	**17682**

Table 5.9: The cut values for rFM and gradient rFM for $K = 256$.

Chapter 6

Conclusion and Future Work

In this thesis, we proposed a heuristic based solution for the replicated hypergraph partitioning problem. In this problem, the vertices are replicated with respect to given replication amount in order to improve the quality of the partitions. This approach differs from the replication schemes in the VLSI literature in the sense that the replication of a vertex cannot bring any net to the cut, i.e., replication does not have any "side effects" except using more space. Our replication scheme can be applied to the hypergraph models in different areas such as distributed IR and spatial databases. For hypergraph partitioning, multilevel and recursive bipartitioning schemes are utilized. The basic FM heuristic is extended to a version that is capable of replication, called replicated FM which introduced new vertex states and gain update algorithms to support replication and unreplication of vertices. We adopted various well-known concepts to further improve the performance of rFM such as early-exit and gradient methodology. We proposed solutions to the issues encountered while integrating our replication scheme into the multilevel and recursive bipartitioning frameworks. These issues include removal of unnecessary replications and replica selection for nets.

The results show that replication is a valuable method to obtain partitions with better quality and the cutsize of the partitions can greatly be reduced using little amount of replication. The different properties of the hypergraphs such as average net degree have a great impact on our replication scheme as the results

indicate. Generally, our replication scheme works well with the hypergraphs with a low average net degree due to the characteristics of the FM based heuristics. Distribution of the given replication amount among bipartitionings has an important effect on the cutsize. Performing more replication at the deeper levels of the recursion tree is a better replication distribution scheme compared to distributing the given replication amount uniformly among all bipartitions. The rFM with gradient methodology can outperform rFM in certain datasets although rFM generally performs better. This may indicate that rFM with gradient methodology should be used with the hypergraphs that have relatively high average net degrees whereas rFM should be used with the hypergraphs with low average net degrees. Replication generally does not disturb the balance of the partitions. However, as the results reveal, different replication distribution schemes have certain effects on the balance of the partitions.

As future research, we have various ideas to that can further improve the quality of the partitions:

- Different operation selection strategies may be tested for rFM like allowing zero gain replication operations. Such an approach can be beneficial since excess replication may have an effect of uncovering new positive gain operations.
- We plan to try different replica selection strategies. A suitable approach may be forcing the selection of the replicas for nets after each level of the recursion tree. In this way, all replicas' parts can be predetermined and the pins of the unused replicas can be removed.
- We can further improve the balance of the partitions by using the remaining replication amount after obtaining a K-way partitioning. This can be achieved by replicating vertices from the most heavily loaded part to other parts.
- A totally different approach would be using a replication scheme that operates on K-way partitions and using this scheme in each level of the recursion tree. In other words, we can use a K-way refinement heuristic that can

perform replication and unreplication operations on vertices. Performing replication in each bipartitioning has the shortcoming of having a global view over the partitions. Even if we obtain bipartitions with a cutsize value of zero, this does not guarantee that the cutsize will be zero after obtaining a K-way partitioning. Therefore, a K-way refinement heuristic can be a perfect tool to overcome this problem.

Bibliography

[1] C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *DAC '97: Proceedings of the 34th annual Design Automation Conference*, pages 530–533, New York, NY, USA, 1997. ACM.

[2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: The VLSI Journal*, 19:1–81, 1995.

[3] C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. *J. Parallel Distrib. Comput.*, 68(5):609–625, 2008.

[4] C. Aykanat, A. Pinar, and U. V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM J. Sci. Comput.*, 25(6):1860–1879, 2004.

[5] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[6] P. A. Bernstein and N. Goodman. The failure and recovery problem for replicated databases. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 114–122, New York, NY, USA, 1983. ACM.

[7] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[8] F. Brglez and B. Zajc. Multi-way netlist partitioning into heterogeneous fpgas and minimization of total device cost and interconnect. In *of Total Device Cost and Interconnect, Design Automation Conference*, pages 238–243, 1994.

[9] T. Brinkhoff. A framework for generating network-based moving objects. *Geoinformatica*, 6(2):153–180, 2002.

[10] T. N. Bui and C. Jones. A heuristic for reducing fill-in in sparse matrix factorization. In *PPSC*, pages 445–452, 1993.

[11] F. Cacheda, V. Plachouras, and I. Ounis. Performance analysis of distributed architectures to index one terabyte of text. *Advances In Information Retrieval, Proceedings*, pages 394–408, 2004.

[12] B. Cahoon and K. S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 110–118, New York, NY, USA, 1996. ACM.

[13] B. B. Cambazoglu and C. Aykanat. A term-based inverted index organization for communication-efficient parallel query processing. *IFIP International Conference on Network and Parallel Computing*, 2006.

[14] U. Catalyurek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.

[15] U. V. Çatalyürek and C. Aykanat. Patoh: partitioning tool for hypergraphs. Technical report, Department of Computer Engineering, Bilkent University, 1999.

[16] U. S. Census Bureau. Topologically integrated geographic encoding and referencing system, 2002. `http://www.census.gov/geo/www/tiger/`.

[17] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, A. Paepcke, S. Raghavan, and G. Wesley. Stanford webbase components and applications. *ACM Trans. Internet Technol.*, 6(2):153–186, 2006.

[18] A. Dasdan and C. Aykanat. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Transactions on Computer-Aided Design*, 16:169–178, 1997.

[19] T. A. Davis. University of florida sparse matrix collection. *NA Digest*, 92, 1994.

[20] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM.

[21] E. Demir, C. Aykanat, and B. B. Cambazoglu. A link-based storage scheme for efficient aggregate query processing on clustered road networks. *Information Systems*, 33(1):1–17, 2008.

[22] E. Demir, C. Aykanat, and B. B. Cambazoglu. A link-based storage scheme for efficient aggregate query processing on clustered road networks. *Inf. Syst.*, 35(1):75–93, 2010.

[23] U. S. Department of Transportation Federal Highway Administration The National Highway Planning Network, 2004. `http://www.fhwa.dot.gov/planning/nhpn/`.

[24] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt. Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large markov models. *J. Parallel Distrib. Comput.*, 64(8):908–920, 2004.

[25] S. Dutt and W. Deng. A probability-based approach to vlsi circuit partitioning. In *DAC '96: Proceedings of the 33rd annual Design Automation Conference*, pages 100–105, New York, NY, USA, 1996. ACM.

[26] M. Enos, S. Hauck, and M. Sarrafzadeh. Evaluation and optimization of replication algorithms for logic bipartitioning. In *In IEEE Transactions on Computer Aided Design*, pages 1237–1248, 1999.

[27] H. Ferhatosmanoğlu, A. c. Tosun, and A. Ramachandran. Replicated declustering of spatial data. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 125–135, New York, NY, USA, 2004. ACM.

[28] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th Design Automation Conference*, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[29] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 173–182, New York, NY, USA, 1996. ACM.

[30] R. H. Güting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[31] L. W. Hagen, D. J. h. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. In *Proc. European Design Automation Conference*, pages 144–149, 1997.

[32] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 28, New York, NY, USA, 1995. ACM.

[33] J. Hwang and A. El Gamal. Optimal replication for min-cut partitioning. In *ICCAD '92: Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*, pages 432–435, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[34] J. Hwang and A. Gamal. Min-cut replication in partitioned networks. *IEEE Trans. on CAD*, 14(1):96–106, Jan 1995.

[35] B.-S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(2):142–153, 1995.

[36] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *DAC '99: Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 343–348, New York, NY, USA, 1999. ACM.

[37] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[38] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J.*, 49:291–307, 1970.

[39] M. Koyutürk and C. Aykanat. Iterative-improvement-based declustering heuristics for multi-disk databases. *Inf. Syst.*, 30(1):47–70, 2005.

[40] C. Kring and A. Newton. A cell-replicating approach to minicut-based circuit partitioning. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 2 –5, 11-14 1991.

[41] B. Krishnamurthy. An improved min-cut algonthm for partitioning vlsi networks. *IEEE Trans. Comput.*, 33(5):438–446, 1984.

[42] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, 10(4):360–391, 1992.

[43] T. Lengauer. *Combinatorial algorithms for integrated circuit layout.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[44] D.-R. Liu and M.-Y. Wu. A hypergraph based approach to declustering problems. *Distrib. Parallel Databases*, 10(3):269–288, 2001.

[45] L.-T. Liu, M.-T. Kuo, C.-K. Cheng, and T. Hu. A replication cut for two-way partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(5):623 –630, May 1995.

[46] L.-T. Liu, M.-T. Kuo, S.-C. Huang, and C.-K. Cheng. A gradient method on the initial partition of fiduccia-mattheyses algorithm. In *ICCAD '95:*

Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design, pages 229–234, Washington, DC, USA, 1995. IEEE Computer Society.

[47] Z. Lu and K. S. McKinley. Partial collection replication versus caching for information retrieval systems. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 248–255, New York, NY, USA, 2000. ACM.

[48] Z. Lu and K. S. McKinley. Partial collection replication for information retrieval. *Inf. Retr.*, 6(2):159–198, 2003.

[49] A. MacFarlane, J. A. McCann, and S. E. Robertson. Parallel search using partitioned inverted files. In *SPIRE '00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, page 209, Washington, DC, USA, 2000. IEEE Computer Society.

[50] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 348–355, New York, NY, USA, 2006. ACM.

[51] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Inf. Retr.*, 10(3):205–231, 2007.

[52] J. A. Orenstein. Redundancy in spatial databases. *SIGMOD Rec.*, 18(2):295–305, 1989.

[53] P. Rigaux, M. Scholl, and A. Voisard. *Spatial databases with application to GIS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[54] Webbase project, `www-diglib.stanford.edu/testbed/doc2/WebBase`.

[55] Y. G. Saab. A fast and robust network bisection algorithm. *IEEE Trans. Comput.*, 44(7):903–913, 1995.

[56] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[57] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *PDIS '93: Proceedings of the second international conference on Parallel and distributed information systems*, pages 8–17, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[58] A. Tomasic and H. Garcia-Molina. Performance issues in distributed shared-nothing information-retrieval systems. *Inf. Process. Manage.*, 32(6):647–665, 1996.

[59] B. Uçar and C. Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM J. Sci. Comput.*, 25(6):1837–1859, 2004.

[60] B. Uçar and C. Aykanat. Revisiting hypergraph models for sparse matrix partitioning. *SIAM Rev.*, 49(4):595–603, 2007.

[61] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *ICDCS '00: Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, page 464, Washington, DC, USA, 2000. IEEE Computer Society.

[62] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, 1997.

[63] H. H. Yang and D. F. Wong. New algorithms for min-cut replication in partitioned circuits. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 216–222, Washington, DC, USA, 1995. IEEE Computer Society.

[64] V. Yazıcı. Balance preserving min-cut replication set for k-way hypergraph partitioning. Master's thesis, Bilkent University, 2010.