5-1-1999

Evaluation of Heuristics in a Distributed Data Staging Network

Noah B. Beck
Purdue University School of Electrical and Computer Engineering

Mitchell D. Theys
Purdue University School of Electrical and Computer Engineering

Howard Jay Siegel
Purdue University School of Electrical and Computer Engineering

Michael Jurczykt
University of Missouri - Columbia, Department of Computer Engineering and Computer Science

Evaluation of Heuristics in a Distributed Data Staging Network

Noah B. Beck[†], Mitchell D. Theys[†],

Howard Jay Siegel[†], and Michael Jurczyk[‡]

[†]Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
{noah, theys, hj)@purdue.edu

[‡]University of Missouri - Columbia
Department of Computer Engineering and Computer Science
Columbia, MO 65211 USA
mjurczyk@cecs.missouri.edu

May 1999

Purdue University
School of Electrical and Computer Engineering
Technical Report TR-ECE-99-7

This research was supported by the DARPA/ISO BADD Program and the Office of Naval Research under ONR Grant N00014-97-1-0804. M. D. Theys was also supported by an Intel Fellowship and an AFCEA Fellowship. Some equipment used in this research was supported by NSF under Grant CDA-9015696, and some was donated by Intel and Microsoft.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Providing up-to-date input to users' applications is an important data management problem for a distributed computing environment, where each data storage location and intermediate node may have specific data available, storage limitations, and communication links available. Sites in the network request data items and each request has an associated deadline and priority. In a military situation, the data staging problem involves positioning data for facilitating a faster access time when it is needed by programs that will aid in decision making. This work concentrates on solving a basic version of the data staging problem in which all parameter values for the communication system and the data request information represent the best known information collected so far and stay fixed throughout the scheduling process. The network is assumed to be oversubscribed and not all requests for data items can be satisfied. A mathematical model for the basic data staging problem is reviewed. Then, three multiple-source shortest-path algorithm based procedures for finding a near-optimal schedule of the communication steps for staging the data are described. Each procedure can be used with each of seven cost criteria developed. A subset of the 21 possible resulting heuristics that are expected to perform well (based on earlier experiments) are evaluated in simulation studies considering different priority weightings schemes, different average number of links used to satisfy each data request, and different network loadings. Finally, an approach considering data items with "more desirable" and "less desirable" available versions is evaluated using a variable time, variable accuracy algorithm, and simulation results are presented. The proposed heuristics are shown to perform well with respect to upper and lower bounds. Furthermore, the heuristics using a complex cost criterion allow more highest priority messages to be received than a simple-cost-based heuristic that schedules all highest priority messages first.

1. Introduction

The DARPA Battlefield Awareness and Data Dissemination (BADD) program [Roc96] includes designing an information system for forwarding (staging) data to proxy servers prior to their usage as inputs to a local application in a distributed computing environment, using satellite and other communication links. The network combines terrestrial cable and fiber with commercial VSAT (very small aperture terminal) internet and commercial broadcast. This provides a unique basis for information management. It will allow web-based information access and linkage as well as server-to-server information linkage. The focus is on providing the ability to operate in a distributed server-server-client environment to optimize information currency for many critical classes of information.

Data staging is an important data management problem that needs to be addressed by the BADD program. A simplified informal description of an example of a data staging problem in a military application is as follows. A warfighter is in a remote location with a portable computer and needs data as input for a program that plans troop movements. The data can include detailed terrain maps, enemy locations, troop movements, and current weather predictions. The data will be available from Washington D.C., foreign military bases, and other data storage locations. One such environment is illustrated in Figure 1.1. Each location may have specific data available, storage limitations, and communication links. Also, each data request is associated with a specific deadline and priority. Depending on the particular environment, there may be hundreds of warfighters, all making multiple requests. It is assumed that not all requests can be satisfied by their deadline. In a military situation, the data staging problem involves positioning data for facilitating a faster access time when it is needed by programs that will aid in decision making.

Positioning the data before it is needed can be complicated by: the dynamic nature of

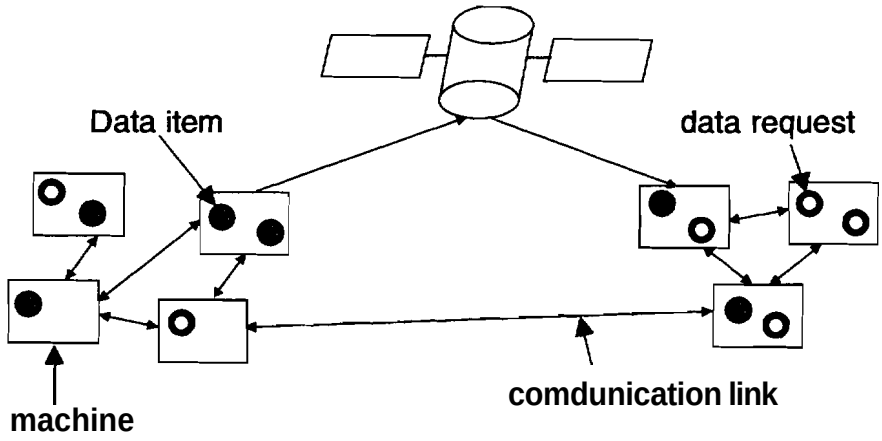

Figure 1.1: An illustration of a data staging environment. Rectangles represent machines, directed lines represent communication links, filled circles represent data items, and open circles represent data requests.

data requests and network congestion; the limited storage space at certain sites; the limited bandwidth of links; the changing availability of links and data; the time constraints of the needed data; the priority of the needed data; and the determination of where to stage the data [Sma96]. Also, the associated garbage collection problem (i.e., determining which data will be deleted or reverse deployed to rear-sites from the forward-deployed units) arises when existing storage limitations become critical [Roc96, Sma96]. The storage situation becomes even more difficult when copies of data items are allowed to reside on different machines in the network so that there are more available sources from which the requesting applications can obtain certain data (e.g., [TaS97, TaS98]). The multiple copies provide an increased level of fault tolerance, in cases of links or storage locations going off-line, and allow the scheduler to select from among different sources to satisfy a data request.

The simplified data staging problem addressed here requires a schedule for transmitting data between pairs of nodes in the corresponding communication system for satisfying as many of the data requests as possible. Each node in the system can be: (a) a source machine of initial data items; (b) an intermediate machine for storing data temporarily; and/or (c) a final destination machine that requests a specific data item.

It is also assumed in this simplified model of the data staging problem that all parameter values for the communication system and the data request information (e.g., network configuration and requesting machines) represent the best known information collected so far and stay fixed throughout the scheduling process. It is assumed that not all of the requests can be satisfied due to storage capacity and communication constraints. The model is designed to create a schedule for movement of data from the source of the data to a "staged" location for the data. It is assumed that a user's application can easily retrieve the data from this location.

Three multiple-source shortest-path algorithm based procedures for finding a near-optimal schedule of the communication steps for staging the data are described [ThT99]. Each procedure can be used with each of seven cost criteria developed. A subset of fourteen of the possible 21 resulting heuristics that are expected to perform well (based on experiments in [ThT99]) are examined in simulation studies considering different priority weighting schemes, different average number of links used to satisfy each data request, and different network loadings. The rationale for considering each of these procedures and costs is provided. he proposed heuristics are shown to perform well with respect to upper and lower bounds. Furthermore, the heuristics using a complex cost criterion are shown to allow more highest priority messages to be received than a simple-cost-based heuristic that schedules all highest priority messages first. Finally, an approach considering data items with "more desirable" and "less desirable" available versions is evaluated using a variable time, variable accuracy algorithm, and simulation results are presented. This research serves as a necessary step toward solving the more realistic and complicated version of the data staging problem involving fault tolerance, dynamic changes to the network configuration, ad hoc data requests, sensor-triggered data transfers, etc.

The material in this report extends the earlier work presented in [ThT99] by introducing three new cost criteria and two new bounds. This work also varies additional simulation parameters, including eight network loadings, three average numbers of links used to get from a source machine to a destination machine, and five priority weighting schemes. This report also introduces a variable time, variable accuracy approach for using data items with "more desirable" and "less desirable" versions.

Section 2 provides an overview of work that is related to the data staging problem. In Section 3, a mathematical model for a basic data staging problem is reviewed. Section 4 provides a detailed description of Dijkstra's algorithm used to find paths of links for transferring data items within the presented network model. Section 5 presents seven cost criteria for use in conjunction with different resource allocation procedures. Three multiple-source shortest-path algorithm based procedures for finding a near-optimal schedule of the communication steps for data staging are described in Section 6. These heuristics adopt the simplified view of the data staging problem described by the mathematical model. Three upper bounds and three lower bounds used to evaluate the performance of these heuristics are presented in Section 7. The set of simulation studies given in Section 8 were created after studying the results of [ThT99]. These new simulation studies examine the effects of (1) having six priority levels with five different weighting schemes, (2) varying the average number of links required for a data item to reach a destination from its source, and (3) varying the total number of requests that must be scheduled in a given network. In Section 9, an approach considering data items with "more desirable" and "less desirable" available versions is evaluated using a variable time, variable accuracy algorithm, and simulation results are presented.

Material in Section 2, parts of Sections 3, 4, 5, 6, and 7 are based on [ThT99]. This material is needed background for the results presented in Sections 8 and 9, and is included here so that this report is self-contained.

A glossary at the end of this report summarizes the terminology used. The source code developed for all of the simulation studies presented here is contained in [Bec99].

2. Related Work*

To the best of the author's knowledge, there is currently no other work presented in the open literature that addresses this version of data staging problem, designs a mathematical model to quantify it, and presents a heuristic for solving it. A problem that is, at a high level, remotely similar to data staging is the facility location problem in management science and operations research [HuM89]. Under the context of the construction of several new production facilities, a manufacturing firm needs to arrange the locations of the facilities and plants effectively, such that the total cost of transporting individual components from the inventory facilities to the manufacturing plants for assembly is minimized. It is required that the firm makes several interrelated decisions: how large and where should the plants be, what production method should be used, and where should the facilities be located. If an analogy is made between: **(1)** the plants and the destination nodes that make the data requests; **(2)** the individual manufacturing components and the requested data elements to be transferred; and **(3)** the production facilities and the source locations of requested data, then at a high level the facility location problem has features similar to those of the data staging problem (e.g., the use of a graph-based method to reduce the facility location problem to a shortest path or minimum spanning tree problem).

However, when examining the relationship between the facility location problem and the data staging problem carefully, there are significant differences. First, each component that a plant requests is usually not associated with a prioritizing scheme, while in the data staging problem each data request has an individual priority. Also, each component requested from a plant commonly does not have a corresponding individual deadline related factor, while in the data staging problem each data request has a deadline. For the data staging problem, the individual priority and individual deadline associated with

*This section is from [ThT99], and is included here so that this report is self-contained.

each data request are the two most important parameters for formulating the optimization criterion. For example, the minimization of the sum of the weighted priorities of satisfiable data requests (based on their individual deadlines) is used as the optimization criterion in the mathematical model of the basic data staging problem presented in Section 3. But for the facility location problem, in general, researchers adopt optimization criteria that are related to the physical distances between plants and facilities in either a continuous or discrete domain without any prioritizing schemes or individual deadline related factors (e.g., [ChD81, CoN80, JoL95, MoC84, Shi77]). Furthermore, in the facility location problem all constraints must be satisfied for the production to occur (e.g., all parts of a car must arrive). In this research, it is known that not all requests can be satisfied (e.g., some low priority data requests may be dropped). Thus, although lessons can be drawn from the design of algorithms for different versions of the facility location problem, there are significant differences between the facility location and the data staging problems in terms of their formulations and potential solutions.

Data management problems similar to data staging for the BADD program are studied for other communication systems. With the increasing popularity of the World Wide Web (WWW), the National Science Foundation (NSF) recently projected that new techniques for organizing cache memories and other buffering schemes are necessary to alleviate memory and network latency and to increase effective bandwidth [Bes97]. More advanced approaches of directory services, data replication, application-level naming, and multicasting are being studied to improve the speed and robustness of the WWW [BaB97]. Evidence has been shown that several file caches could reduce file transfer traffic, and hence the volume of traffic on the internet backbone [DaH93]. In addition, ways to increase distributed system performance with intelligent data placement have been studied [AcZ93]. The study of data staging can potentially draw lessons from and generate positive input for the active research in these related, but not directly comparable, areas.

Work has been done to provide extensions to wormhole routing protocols that handle real-time messages. An off-line approach that schedules usage of the virtual channels by allowing higher priority messages to preempt lower priority messages is presented in

[BaO98]. Their research shows that they improve wormhole routing by employing such a protocol. The goal of the work in [BaO98] is similar to the goal of the work presented here in that both give preference to messages that have higher priority. However, in [BaO98] the focus is on wormhole routing protocols, while the work presented here (1) is for a general communication system; (2) attempts to find minimum paths over multiple links; and **(3)** uses a cost criterion that also considers how close a message is to its deadline.

There has been research done in the area of mapping tasks onto a suite of distributed heterogeneous machines (e.g., [BrS98, BrS99, HeK99, MaA99, WaS97]). This task mapping research focuses on deciding what machine should execute each task, rather than assuming the task execution locations are known (as in the data staging situation). Thus, the basic problem being addressed by these task mapping studies is different than that of data staging.

Other research exploring heuristics for use in the BADD environment has been performed [LeB97]. This work examines methods for scheduling efficiently the ATM-like channels of a possible BADD-like environment. It shows that "greedy" heuristics are effective tools for use in that BADD-like environment and uses a network simulator to corroborate this statement; however, those heuristics do not consider several parameters considered here, such as deadlines and data availability times. The work here differs from [LeB97] in that: (1) here a detailed mathematical model is developed, and (2) the collection of heuristics and cost criteria studied here are based on a differentset of assumptions about system structure and data request characterizations.

3. Mathematical Data Staging Model*

3.1 Model Definition

Consider a network topology graph G_{nt} composed of a set of vertices that represent the set of machines $\underline{M}$ in the network and a set of communication links $\underline{L}$ that represent the directed edges. There are $\underline{m}$ machines in M, identified as $\{M[0], M[1], \ldots, M[m-1]\}$, and each can be a source, destination, and intermediate location for data items in the network. Source machines for data items are the machines where data items are initially located within the network; these data items may eventually be transferred by the network to destination machines, possibly stored at intermediate machines along the way. Each machine $\underline{M[i]}$ (where $0 \leq$ i $<$ m) also has an associated constant unused storage capacity during the time interval $[t_j, t_{j+1})$, $\underline{Cap[i](t_j)}$. Note that the times t_j and t_{j+1} may not differ by exactly one time unit.

Communication links in this system are represented as one or more virtual links. A virtual link corresponds to a period of constant, continuous, available bandwidth from one machine to one other machine. Bidirectional communication links are therefore represented as two virtual links-one for each direction. A communication link that is only available during certain time intervals is represented by a separate virtual link for each period of availability. $\underline{Nl[i, \mathrm{j}]}$ is the number of virtual links from machine $M[i]$ to $M[j]$ (where $i \neq$ j and $0 \leq$ i, j $<$ m). The kth virtual link from machine $M[i]$ to $M[j]$ is identified as $\underline{L[i,j][k]}$ (where $0 \leq$ k $< Nl[i, \mathrm{j}]$). The virtual link $L[i,j][k]$ also has an associated link starting time $\underline{Lst[i,j][k]}$, denoting the time when it becomes available, as well as a link ending time $\underline{Let[i,j][k]}$, which specifies the time when the link is no longer available.

'This model is based on the one in [ThT99], which builds on [TaT98]. It has been modified to include multiple versions of a given data item. This material is needed background for the results presented in Sections 8 and 9, and is included here so that this report is self-contained.

Data items are blocks of information that can be transmitted from one machine to another. The set of data items with unique names or identifiers that are available on the machines in M is called Δ. Names or identifiers assigned to data items must be different if the contents of the data items are different in any way, including details such as differing timestamps on weather maps of the same region. The number of distinctive data items in **A** is n, and individual unique data items are identified as $\{\delta[0], \delta[1], \ldots, \delta[n-1]\}$. For a data item $\delta[l]$ (where $0 \leq l <$ n), the size of the data item is represented as $|\delta[l]|$. The time duration required to transfer data item $\delta[l]$ from machine $M[i]$ to machine $M[j]$ (where $i \neq$ j and $0 \leq i, \mathrm{j} <$ m) via the virtual link $L[i,j][k]$ (where $0 \leq$ k $< Nl[i, \mathrm{j}]$) during the time interval $[Lst[i,j][k], Let[i, \mathrm{j}][k]]$ is $D[i,j][k](|\delta[l]|)$. Machine $M[i]$ may be a source of $\delta[l]$, or an intermediate storage location or destination that already holds a copy of $\delta[l]$. Machine $M[j]$ may be an intermediate storage location or a destination.

Let $N\delta[l]$ (where $0 \leq 1 <$ n) represent the number of source machines holding a copy of $\delta[l]$, and $M[Source[l, \mathrm{j}]]$represent the jth source machine for data item $\delta[l]$ (where $0 \leq \mathrm{j} < N\delta[l]$ and $0 \leq Source[l, \mathrm{j}] <$ m). The starting time $\delta st[l,j]$ refers to the time data item $\delta[l]$ becomes available at its jth source machine. The removal time $\delta rt[l, \mathrm{i}]$ (where $0 \leq i <$ m) refers to the time data item $\delta[l]$ can be removed from machine $M[i]$, if a copy of $\delta[l]$ is being stored at $M[i]$. This allows the value of $Cap[i](\delta rt[l, \mathrm{i}])$to be increased by $|\delta[l]|$. Intermediate machines, for example, could set $\delta rt[l, \mathrm{i}]$ to be some small time period α after the last deadline at any machine for data item $\delta[l]$. This would allow the storage space to be reclaimed at intermediate machines after the usefulness of the data item has expired. The scheduling heuristics do not remove a data item from any of its sources or destinations because this is considered outside the scope of responsibility of the scheduler.

Consider now a data item such as an image showing a map of a planned battle area. It may be possible to have available a higher quality version of the image that shows a higher level of detail, as well as a lower quality version showing less detail. A person requesting this data item would prefer to receive the higher quality image, but it may be that there are not enough resources (e.g., network bandwidth) available to fulfill this data request. In this event, however, there may be enough resources available to send

the lower quality image, which would be better than sending nothing at all.

The set Rq (where Rq $\subseteq$ **A)** contains unique data items requested by destination machines in M. The number of unique data items in Rq is 2ρ; the higher quality data items are identified as $\{Rq[0], Rq[1], \ldots, Rq[\rho - 1]\}$, and the lower quality data items are identified as $\{Rq[\rho], Rq[\rho + 1], \ldots, Rq[2\rho - 1]\}$. Here, each requested higher quality data item $Rq[i]$ (where $0 \leq i <$ p) has a corresponding lower quality data item $Rq[i +$ p] that may be sent in place of $Rq[i]$ if system resources become limited. Note that for every i there must exist exactly one j and exactly one k such that $Rq[i] = \delta[j]$ and $Rq[i + \rho] = \delta[k]$. These data items $\delta[j]$ and $\delta[k]$ are assumed for simplicity to be present at the same source machines, and to have the same associated starting times and removal times. This model also assumes for simplicity that $|Rq[i + \rho]| = \frac{1}{2}|Rq[i]|$.

The number of destination machines that request $Rq[i]$ (where $0 \leq$ i $<$ p) is denoted with $Nrq[i]$. If $0 \leq$ k $< Nrq[i]$, then $M[Request[i,$ k$]]$ refers to the kth machine that requested $Rq[i]$ (where $0 \leq Request[i,$ k$] <$ m). Each of these machines also implicitly requests $Rq[i +$ p] in the event that $Rq[i]$ cannot be sent, so that $Nrq[i + \rho] = Nrq[i]$, and $Request[i +$ p, k$] = Request[i,$ k] for all values of k. The finishing time $Rft[i,$ k] (and equivalent $Rft[i +$ p, k]) refers to a deadline time, after which data item $Rq[i]$ (and $Rq[i +$ p]) is no longer useful to machine $M[Request[i,$ k]]. The requesting machine M[Request[i, k]] also associates the data item Rq[i] (and corresponding $Rq[$i$+$ p]) with a numbered priority class $Priority[i,$ k] (equal to $Priority[i + \rho,$ k]). The highest, or most important priority class is P, and the lowest, or least important priority class is 0, so that $0 \leq Priority[i,$ k$] \leq$ P .

Define a schedule as a series of communication steps, among the machines of M using the communication links in L, that transfer some or all of the data items in the set Rq from their respective source machines to some or all of their respective destination machines, possibly being stored at intermediate machines along the way. Suppose that there are σ possible distinct schedules, enumerated $\{S_0, S_1, \ldots, S_{\sigma-1}\}$. The kth (where $0 \leq$ k $< Nrq[j]$) request for a data item $Rq[j]$ (where $0 \leq$ j $< 2\rho$) is considered satisfiable with respect to a specific schedule S_h (where $0 \leq$ h $<$ a) if and only if the data item $Rq[j]$ is available at machine $M[Request[j,$ k$]]$ at or before the deadline time $Rft[j,$ k].

The set $\underline{Srq[S_h]}$ then denotes the set of two-tuples (j,k) such that the kth request for the data item $Rq[j]$ is satisfiable with respect to the schedule S_h.

There must be a way to represent the relative importance of a priority class *a:* (where $0 \leq \alpha \leq$ P) compared to another priority class β (where $0 \leq \beta \leq$ P and *a:* $\neq \beta$). The relative weight of any priority class *a:* is denoted by $\underline{W[\alpha]}$. This means that if priority class *a:* is ten times as important as priority class β, the value of $W[\alpha]$ will be ten times the value of W[β].

Let $\underline{Worth[j, \mathrm{k}]}$ (where $0 \leq \mathrm{j} < 2\rho$ and $0 \leq \mathrm{k} < Nrq[j]$) denote a percentage of value to a user of data item $Rq[j]$ sent to satisfy a request at machine $M[Request[j, \mathrm{k}]]$. For simplicity, this model assumes that if $Rq[i]$ for $0 \leq \mathrm{i} < \rho$ is sent to $M[Request[i, \mathrm{k}]]$ by its deadline, then $Worth[i, \mathrm{k}] = 1$ (meaning 100% for the preferred data version), and $Worth[i+\rho, \mathrm{k}] = 0$ (meaning no additional worth for the second data version). If $Rq[i]$ is not sent to $M[Request[i, \mathrm{k}]]$ by its deadline, and $Rq[i+\rho]$ is sent to $M[Request[i + \mathrm{p}, k]]$ by its deadline, then $Worth[i + \mathrm{p}, \mathrm{k}] = 0.25$ (meaning 25% for the lower quality version), and $Worth[i, \mathrm{k}] = 0$. Now, the effect of the schedule S_h (where $0 \leq \mathrm{h} < a$) can be defined

$$\underline{E[S_h]} = -\left(\sum_{(j,k) \in Srq[S_h]} W[Priority[j,k]] * Worth[j, \mathrm{k}] \right)$$

(where $0 \leq \mathrm{j} < 2\rho$ and $0 \leq \mathrm{k} < Nrq[j]$). The global optimization criterion, and hence, the objective of all of the heuristics presented later, is to find the schedule with the minimum effect, defined as

$$\min_{0 \leq h < \sigma} E[S_h].$$

Another way to view this minimization is to think of it as trying to find the schedule of data transfers that produces the maximum sum of satisfied requests' priority weights.

3.2 Heuristic Solution Approach

The heuristic approach used in this report to create the schedule S_h with minimum effect $E[S_h]$ utilizes Dijkstra's shortest path algorithm. This algorithm, presented in Section **4,** calculates arrival times for data items and establishes paths of virtual links to get data items from source machines to destination machines. The paths calculated by this algorithm give the earliest arrival time for a given data item, provided that

there are no other data items competing for resources in the network. After Dijkstra's algorithm has been run for each requested data item (i.e., all data items in Rq), a single data item and one or more destination machines are selected through the use of a cost criterion presented in Section 5. This data item choice reflects a combination of its contribution to the effect of the schedule, and the amount of time between its arrival at a destination and its deadline at that destination. Network resources and machine storage are then allocated according to one of the procedures presented in Section 6, updating link availability times and available machine storage. This updating of network information will cause the arrival times and virtual link paths for some other data items to become invalid, so the heuristic process (using a cost and an allocation procedure) is repeated again (beginning with Dijkstra's algorithm) using the modified network information. This continues until there are no more satisfiable data items in the network, thus producing the communication schedule. Results from simulation studies using this approach, which only considers one version of each data item (i.e., considers only $Rq[i]$ where $0 \leq i < \rho$, not $Rq[j]$ where $p \leq j < 2\rho$), are found in [ThT99] and Section 8. A modified approach considering both versions of a data item is contained in Section 9.

4. Dijkstra's Shortest Path Algorithm*

The heuristics presented here utilize Dijkstra's algorithm [CoL90] for finding the shortest path from one or more source nodes to all other nodes in a directed graph. The version used calculates the earliest possible available time for a data item $Rq[i]$ (where $0 \leq \mathrm{i} < 2\rho$) at each machine in M, given a subset of machines in M that already holds a copy of $Rq[i]$.

Define the available time $A_T[i, \mathrm{j}]$ (where $0 \leq i <$ 2p, $0 \leq \mathrm{j} <$ m) as the earliest possible time found so far when data item $Rq[i]$ could be present and available at machine $M[j]$. Define also the value of the predecessor $\pi[i, \mathrm{j}]$ to be the two-tuple (s,k) (where $-1 \leq \mathrm{s} <$ m, $-1 \leq \mathrm{k} < Nl[s, \mathrm{j}]$) identifying the machine $M[s]$ as the machine that sends data item $Rq[i]$ to machine $M[j]$ via virtual link $L[s,j][k]$. If the value of $\pi[i,j]$ is $(-1,-1)$, this means that no machine sends data item $Rq[i]$ to machine $M[j]$ via any virtual link. This may happen if machine $M[j]$ is a source machine for data item $Rq[i]$, or it may happen if it is not possible for machine $M[j]$ to receive a copy of data item $Rq[i]$ (possibly due to the unavailability of network resources).

The pseudocode for Dijkstra's algorithm is shown in Figure 4.1. This algorithm is invoked once for each data item in Rq to establish an available time and predecessor on each machine for each data item. The pseudocode applies the algorithm for $Rq[i_{Rq}]$, which corresponds to $\delta[i_\delta]$.

As an example, consider the machines and virtual links shown in Figure 4.2. In this figure, Dijkstra's algorithm will be applied for data item $Rq[0]$, which in **A** corresponds to $\delta[0]$ (i.e., $i_{Rq} = 0$ and $i_\delta = 0$ for this example). Machine $M[0]$ is the only source machine for this data item ($N\delta[0] = 1$ and $Source[0,0] = 0$), and the data item becomes present at $M[0]$ at time $\delta st[0,0] = 0$. Step 1 of the algorithm finds the index i_δ of data item $Rq[0]$ in the **A** set ($i_\delta = 0$) and can be implemented as a simple table lookup operation. Then,

'This section is based on [ThT99]. This material is needed background for the results presented in Sections 8 and 9, and is included here so that this report is self-contained.

DIJKSTRA(M, δst, $N\delta$, i_{Rq}, Nl, Lst, Let, D, A_T, π)

1. assign i_δ such that $\delta[i_\delta] = Rq[i_{Rq}]$
2. for all $j \in \{0, 1, \ldots, m-1\}$
3. $\quad \pi[i_{Rq}, j] \leftarrow (-1, -1)$
4. $\quad A_T[i_{Rq}, j] \leftarrow \infty$
5. for all $j \in \{0, 1, \ldots, N\delta[i_\delta] - 1\}$
6. $\quad A_T[i_{Rq}, Source[i_\delta, j]] \leftarrow \delta st[i_\delta, j]$
7. $M_U \leftarrow M$
8. while $M_U \neq \{\}$
9. $\quad$ assign j_{min} such that $A_T[i_{Rq}, j_{min}] = \min(A_T[i_{Rq}, j], M[j] \ E \ M_U)$
10. $\quad M_U \leftarrow M_U - M[j_{min}]$
11. $\quad$ for all j such that $M[j] \in M_U$
12. $\quad\quad$ for all $k \in \{0, 1, \ldots, Nl[j_{min}, j] - 1\}$
13. $\quad\quad\quad$ if $A_T[i_{Rq}, j_{min}] < Lst[j_{min}, j][k]$
14. $\quad\quad\quad\quad$ if $Lst[j_{min}, j][k] + D[j_{min}, j][k](|Rq[i_{Rq}]|) \leq Let[j_{min}, j][k]$
15. $\quad\quad\quad\quad\quad$ if $Lst[j_{min}, j][k] + D[j_{min}, j][k](|Rq[i_{Rq}]|) < A_T[i_{Rq}, j]$
16. $\quad\quad\quad\quad\quad\quad$ if $\min(Cap[j](t), Lst[j_{min}, j][k] \leq t \leq \delta rt[i_\delta, j]) \geq |Rq[i_{Rq}]|$
17. $\quad\quad\quad\quad\quad\quad\quad A_T[i_{Rq}, j] \leftarrow Lst[j_{min}, j][k] + D[j_{min}, j][k](|Rq[i_{Rq}]|)$
18. $\quad\quad\quad\quad\quad\quad\quad \pi[i_{Rq}, j] \leftarrow (j_{min}, k)$
19. $\quad\quad\quad$ else if $A_T[i_{Rq}, j_{min}] + D[j_{min}, j][k](|Rq[i_{Rq}]|) < Let[j_{min}, j][k]$
20. $\quad\quad\quad\quad$ if $A_T[i_{Rq}, j_{min}] + D[j_{min}, j][k](|Rq[i_{Rq}]|) < A_T[i_{Rq}, j]$
21. $\quad\quad\quad\quad\quad$ if $\min(Cap[j](t), A_T[i_{Rq}, j_{min}] \leq t \leq \delta rt[i_\delta, j]) \geq |Rq[i_{Rq}]|$
22. $\quad\quad\quad\quad\quad\quad A_T[i_{Rq}, j] \leftarrow A_T[i_{Rq}, j_{min}] + D[j_{min}, j][k](|Rq[i_{Rq}]|)$
23. $\quad\quad\quad\quad\quad\quad \pi[i_{Rq}, j] \leftarrow (j_{min}, k)$
24. return(π, A_T)

Figure *4.1:* Dijkstra's algorithm for finding the earliest available times and links for getting data item $Rq[i_{Rq}]$ from source machines to destination machines.

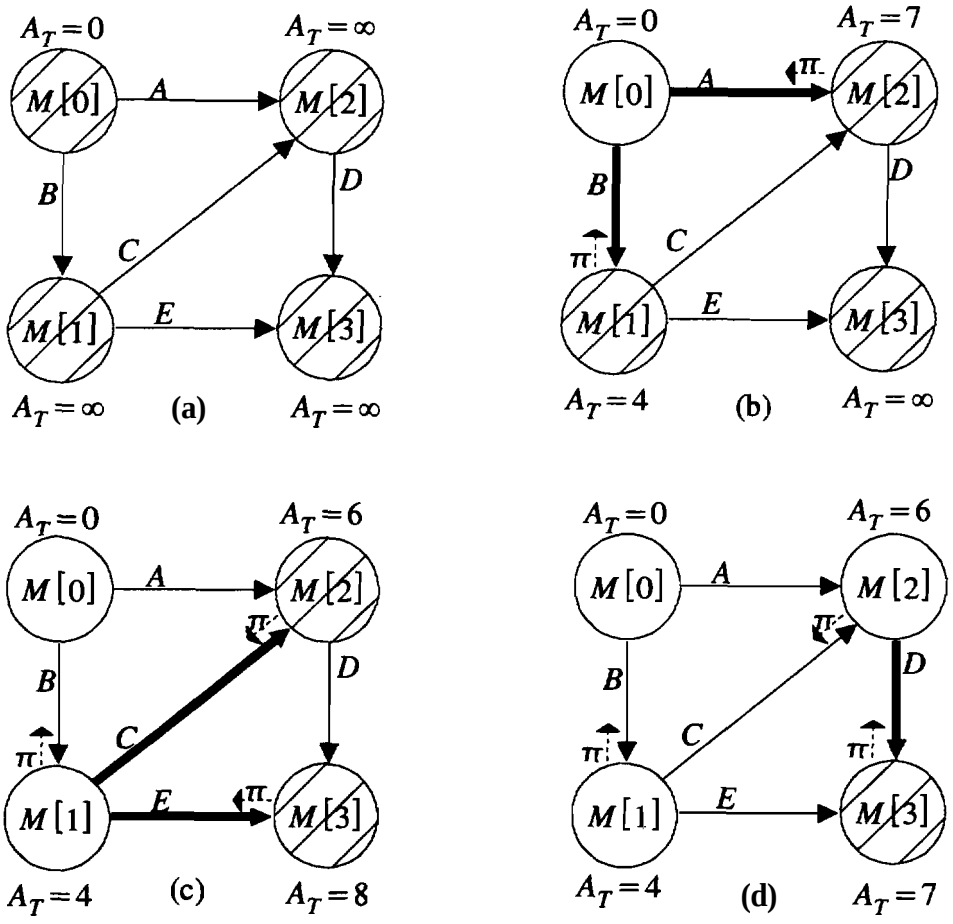

Figure 4.2: An example of Dijkstra's algorithm being applied to a simple network.

the predecessor (π) and available time (A_T) arrays are initialized for data item $Rq[0]$ on each machine in steps 2 through 4. The initial values for A_T indicate that no available time has been found so far for each machine, and the initial values for π indicate that no machine is receiving data item $Rq[0]$ from any other machine. Steps 5 and 6 set the available time (A_T) array elements for machines that are sources for data item $Rq[0]$. This will have the effect of setting $A_T[0,0] = 0$, resulting in the network shown in Figure 4.2(a). The set M_U represents the set of machines for which the earliest possible available time for data item $Rq[i_{Rq}]$ is currently unknown. Step 7 initializes this set to contain all machines in M; machines that are members of M_U are denoted in Figure 4.2 with a diagonal line fill in the machine node.

Steps 8 through 23 of Figure 4.1 constitute the main loop that performs the machine selection and relaxation phases of Dijkstra's algorithm. It continues until all machines have been selected and removed from the set M_U, which implies that the earliest possible available times at all machines are known. The machine selected from M_U in step 9 is the one that has the earliest available time for data item $Rq[0]$, which in this case is the source machine $M[0]$ (meaning j_{min} is set to 0). This machine is then removed from M_U in step 10 before its outgoing links are relaxed beginning in step 11. This relaxation of links in steps 11 through 23 consists of updating the available times at machines that can be improved by using those links.

Steps 11 and 12 iterate through all virtual links that originate from the selected machine and connect to all other machines remaining in M_U. For simplicity, Figure 4.2 shows only virtual links that will enable the relaxation of an available time. Suppose link A represents virtual link $L[0,2][4]$ with a start time of $Lst[0,2][4] = 2$, a finishing time of $Let[0,2][4] = 8$, and where the data transfer duration $D[0,2][4](|Rq[0]|) = 5$. Similarly, suppose link B represents virtual link $L[0,1][3]$ with associated values $Lst[0,1][3] = 3$, $Let[0,1][3] = 8$, and $D[0,1][3](|Rq[0]|) = 1$. For both of these links in step 13, the link starting time (Lst) is after the available time (A_T) at machine $M[0]$, meaning that the earliest that the data item can begin to be transferred is at the link starting time. Both links are available for long enough in step 14 to transfer the entire contents of data item $Rq[0]$.

The new available times in step 15 that would be generated by using these links are less than the current available times of ∞ at machines $M[1]$ and $M[2]$. Step 16 then checks to see if there is enough storage space at the receiving machines to store the data item until the time it will be deleted. For the purpose of simplifying this example, it is assumed that enough storage space is available to store $Rq[0]$ indefinitely at all machines in the network. Steps 17 and 18 then perform the actual relaxation—using $L[0,2][4]$ (link A in Figure 4.2(b)) relaxes $A_T[0,2]$ from ∞ to 7 and sets $\pi[0,2] = (0,4)$, while using $L[0,1][3]$ (link B in Figure 4.2(b)) relaxes $A_T[0,1]$ from ∞ to 4 and sets $\pi[0,1] = (0,3)$. These actions for $M[1]$ and $M[2]$ occur during separate iterations of the loop beginning at step 11. Steps 19 through 23 perform the same functions as steps 14 through 18,

except that they are called for the case where the available time at the machine selected from M_U is after the link starting time for a given link.

The next machine selected in step *9* as $M[j_{min}]$ is $M[1]$. In Figure 4.2(c), it is assumed that *(1)* using link C ($L[1,2][4]$) allows $A_T[0,2]$ to be relaxed from *7* to 6 and changes the predecessor $\pi[0,2]$ to $(1,4)$, and *(2)* link E ($L[1,3][2]$) allows $A_T[0,3]$ to be relaxed from ∞ to *8,* setting $\pi[0,3] = (1,2)$. The next machine selected in step *9* as $M[j_{min}]$ is $M[2]$. In Figure 4.2(d), it is assumed that link D ($L[2,3][1]$) can then be used to relax $A_T[0,3]$ from *8* to *7,* and change $\pi[0,3]$ to $(2,1)$. Finally, machine $M[3]$ is selected in step *9,* but there are no machines remaining in M_U, so the algorithm returns with all of the earliest possible available times in A_T, and predecessor machines in π for each machine that can receive $Rq[0]$.

5. Data Item Selection Cost Criteria*

5.1 Introduction

Network resources must be allocated to data requests in some order; this order intuitively should include "more important" requests and requests that are "close" to their deadlines before "less important" requests and requests that are "not close" to their deadlines. Some quantitative cost must therefore be applied so that an algorithm can evaluate the relative merit of any given request compared to any other request. Seven different cost criteria are detailed below; each attempts to take into consideration both the importance of a data request, and how close the data request is to its deadline.

Suppose $M[r]$ (where $0 \leq$ r $<$ m) is the next machine to receive data item $Rq[i]$ (where $0 \leq$ i $< 2\rho$) on a path from $M[s]$ (where $(s, l) = \pi[i, r]$), which can be any machine already holding a copy of $Rq[i]$, to one or more requesting destination machines. That is, machine $M[s]$ holds a copy of data item $Rq[i]$, and $M[r]$ must be the next machine to receive $Rq[i]$ so that $M[Request[i, k]]$ (for one or more values of k, where $0 \leq k < Nrq[i]$) can ultimately receive $Rq[i]$. Let the set of values of k that satisfy this condition (i.e., destination machines that request $Rq[i]$ through $M[r]$) be called $\underline{Drq[i, r]}$.

Assume that $Rq[i]$ is the next data item to be allocated network resources. Let the value $\underline{Sat[i, k]}$ (where $0 \leq i < 2\rho$ and $0 \leq k < Nrq[i]$) be 1 if $Request[i, k]$ would be satisfiable, and 0 if it would not be satisfiable. For the simulations of [ThT99] and Section 8, $Sat[i, k]$ is 0 for values of i such that $p \leq i < 2\rho$, thus ignoring the less desirable data item versions. Now, the effective priority $\underline{Efp[i, k]}$ of data item $Rq[i]$ at the kth requesting location can be defined as $Sat[i, k] * W[Priority[i, k]] * Worth[i, k]$. An urgency term, indicating how close a data item's available time is to its deadline time (in seconds) at a destination is defined as $\underline{Urgency[i, k]} = -Sat[i, k] * (Rft[i, k] - A_T[i, Request[i, k]] + 1)$.

*Cost criteria $C1$, C2, C3, and C4 were defined in [ThT99]. These criteria are needed background for the results presented in Sections 8 and 9, and are included here so that this report is self-contained.

A smaller urgency here indicates that it is less urgent to get $Rq[i]$ to $M[Request[i, \mathrm{k}]]$. The "+1" in the urgency term is so that the urgency never becomes a small number close to zero.

The next value that must be defined before detailing the cost criteria is the number of virtual links used to get from a machine $M[s]$ (where $(\mathrm{s}, \mathrm{I}) = \pi[i, r]$ and $M[r]$ is the next intermediate machine described above) holding a copy of data item $Rq[i]$ (where $0 \leq i < 2\rho$) to destination machine $M[Request[i, \mathrm{k}]]$ (where $0 \leq \mathrm{k} < Nrq[i]$). Let this value be called $Nlinks[i, \mathrm{k}]$, and note that it reflects the number of links used in the path (generated by the most recent run of Dijkstra's algorithm) from a machine holding the data item to a machine requesting the data item.

All of the following cost functions take into account the priority and urgency of a data item. Six of the costs allow the weight assigned to the priority term to be varied relative to the weight assigned to the urgency term. These weighting terms are W_E for the weight of the effective priority term, and W_U for the weight of the urgency term. The relative weight of these two terms compared to each other (W_E/W_U) is called the E-U ratio. For all cost criteria, a smaller value indicates a more desirable use of communication resources; therefore, resource allocation is performed by the procedures in Section 6 for the data item and destination machine(s) with minimum cost.

5.2 Cost $C1$

The first cost, initially described in [TaT98], is referred to as $C1$. It gives an individual value based on each data item at each requesting destination and does not take into account other destinations requesting the same data item. It quantifies the cost for sending data item $Rq[i]$ (where $0 \leq i < 2\rho$) to $M[r]$ (where $0 \leq \mathrm{r} < m$) from $M[s]$ via link $L[s, \mathrm{r}][\mathrm{k}]$ (where $(\mathrm{s}, \mathrm{k}) = \pi[\mathrm{i}, r]$), in order to ultimately try to satisfy the jth (where $0 \leq \mathrm{j} < Nrq[i]$) requesting destination machine:

$$C1[i, j][s, r][k] = -W_E * Efp[i, j] - W_U * Urgency[i, j]$$

This cost is calculated for all values of i and corresponding values of j, and for values of s, r, and k corresponding to the shortest paths found to each satisfiable destination. The first term in the equation attempts to give preference to a data request with a

priority higher than the other requests. Furthermore, to satisfy as many data requests as possible, intuitively it is necessary to transfer a specific data item to the requesting locations whose deadlines are sooner. This intuition is captured by the inclusion of the urgency term. Thus, with the collective consideration of the priority of satisfiable data requests and the urgency of those data requests in this local optimization step, a near-optimal communication schedule that reasonably achieves the global optimization criterion should be generated.

5.3 Cost C2

This cost criterion $C2$ collectively considers all requesting destination machines that would benefit from sending a data item to a common intermediate storage machine. It quantifies the cost for sending data item $Rq[i]$ (where $0 \leq$ i $< 2\rho$) to $M[r]$ (where $0 \leq$ r $<$ m) from $M[s]$ via link $L[s,r][k]$ (where (s, k) $= \pi[i,$ r]), in order to ultimately try to satisfy the jth (where j $\in Drq[i,$ r]) requesting destination machine(s):

$$C2[i][s,r][k] = -W_E * \left(\sum_{j \in Drq[i,r]} Efp[i,j] \right) - W_U * \left(\max_{j \in Drq[i,r]} Urgency[i,j] \right).$$

Rather than summing all of the urgency terms for the destinations, the most urgent satisfiable request is added in C2. This method of capturing the urgency is used as a heuristic to maximize the sum of the weighted priorities of satisfied requests because if the most urgent request for an item passing through $M[r]$ is satisfied, it is more likely that all requests for this data item passing through $M[r]$ will be satisfied.

5.4 Cost C3

The cost criterion $C3$ also collectively considers all requesting destination machines that would benefit from sending a data item to a common intermediate storage machine. It quantifies the cost for sending data item $Rq[i]$ (where $0 \leq$ i $< 2\rho$) to $M[r]$ (where $0 \leq$ r $<$ m) from $M[s]$ via link $L[s,r][k]$ (where $(s,$ k) $= \pi[i,r]$), in order to ultimately try to satisfy the jth (where $j \in Drq[i,$ r]) requesting destination machine(s):

$$C3[i][s,r][k] = \sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{Urgency[i,j]}.$$

This criterion is a sum of the weighted priorities of satisfiable requests for data item $Rq[i]$ on a path through machine $M[r]$ normalized by the urgency of each request. Note that

this cost does not use W_E or W_U. This is because the effective priority is divided by the urgency and so W_E divided by W_U acts as a scaling factor that would not affect the relative cost of the requests. That is, for two data items $Rq[i_1]$ and $Rq[i_2]$ competing for the use of $L[s,r][k]$, the relative value of $\frac{C3[i_1][s,r][k]}{C3[i_2][s,r][k]}$ will be unchanged by including any given W_E to weight the $Efp[i, \mathbf{j}]$ factors and any given W_U to weight the $Urgency[i, \mathbf{j}]$ factors.

5.5 Cost C4

The cost $\underline{C4}$ for transferring data item $Rq[i]$ (where $0 \leq$ i $< 2\rho$) to $M[r]$ (where $0 \leq$ r $<$ m) from $M[s]$ via link $L[s,r][k]$ (where (s,k) $= \pi[i,$ r$]$),in order to ultimately try to satisfy the jth (where $\mathbf{j} \in Drq[i,$ r$]$) requesting destination machine(s):

$$C4[i][s,r][k] = -W_E * \left(\sum_{j \in Drq[i,r]} Efp[i,j] \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right).$$

This cost sums the weighted priorities of all satisfiable requests for data item $Rq[i]$ on a path through machine M[r]and combines that with the sum of the urgency for those same satisfiable requests. Comparing $C2$ and $C4$, it should be noted that the urgency term for each destination whose shortest path shares an intermediate node $M[r]$ is summed in C4, whereas $C2$ simply takes the maximum of the urgency terms over this same set of destinations. The benefit of C4 is demonstrated by the following example. The first data item, $Rq[i_1]$, is requested by four machines that all have identical priorities, and have an A_T that is very close to their deadlines. The second data item, $Rq[i_2]$, is also requested by four destinations that have the same identical priorities, but only one destination has an A_T that is close to its deadline. $C2$ will be unable to differentiate between these two data requests, but C4 will choose to schedule $Rq[i_1]$ before $Rq[i_2]$.

5.6 Cost *C4links*

Based on $C4$ because of its high performance in simulation tests, cost $\underline{C4links}$ is also defined for transferring data item $Rq[i]$ (where $0 \leq$ i $< 2\rho$) to $M[r]$ (where $0 \leq$ r $<$ m) from M[s]via link $L[s,r]$[k] (where (s,k) $= \pi[i,$ r$]$),in order to ultimately try to satisfy the jth (where $\mathbf{j} \in Drq[i,$ r$]$) requesting destination machine(s):

$$C4links[i][s,r][k] = -W_E * \left(\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{Nlinks[i,j]} \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right).$$

Because a data request that can be satisfied by using three virtual links is using three times as much network resources as a data request that can be satisfied by using only one virtual link, this cost divides the effective priority term for each requesting destination by the number of links used to get to that destination. If the effective priority associated with a data request is considered as a measure of worth or importance to the user, then this first term would be considered a measure of worth per link. This should allow the cost criterion to better select data items to satisfy that will make the most effective use of the network resources available.

5.7 Cost $C4size$

Based again on C4 because of positive simulation results, the criterion $\underline{C4size}$ is also defined for transferring data item $Rq[i]$ (where $0 \leq i < 2\rho$) to $M[r]$ (where $0 \leq$ r $<$ m) from $M[s]$ via link $L[s,r][k]$ (where (s, k) $= \pi[i,$ r]), in order to ultimately try to satisfy the jth (where $\mathbf{j} \in Drq[i,$ r]) requesting destination machine(s):

$$C4size[i][\mathrm{s},r][\mathrm{k}] = -W_E * \left(\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{|Rq[i]|} \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i, \mathbf{j}] \right).$$

A data request with an effective priority p representing its worth to the recipient, and a size in bytes of q, then has an effective worth per byte of $\frac{p}{q}$. Because the goal of a cost criterion is to identify data requests that will make the most effective use of network resources, the first term in $C4size$ uses this effective priority divided by data request size to find data items that will transmit the maximum amount of worth per link bandwidth byte.

5.8 Cost $C4sizlnk$

Cost $\underline{C4sizlnk}$ is a combination of the ideas in $C4size$ and $C4links$, and gives a cost for transferring data item $Rq[i]$ (where $0 \leq i < 2\rho$) to $M[r]$ (where $0 \leq$ r $< m$) from M[s]via link $L[s,$ r][k] (where (s, k) $= \pi[i,$ r]), in order to ultimately try to satisfy the jth (where $\mathbf{j} \in Drq[i,$ r]) requesting destination machine(s):

$$C4sizlnk[i][s,r][k] = -W_E * \left(\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{|Rq[i]| * Nlinks[i,j]} \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right).$$

By combining the size and number of virtual links used, this cost gives a more accurate calculation of the resources used by a data request. For instance, consider two data items $Rq[i_1]$ and $Rq[i_2]$ of equal priority. Consider also that $Rq[i_2]$ is twice as large as $Rq[i_1]$, and that it requires the use of three virtual links versus $Rq[i_1]$'s single virtual link. In this case, $Rq[i_2]$ is requiring six times the total network resources required by $Rq[i_1]$ in order to satisfy the same priority level of request.

6. Resource Allocation Procedures*

6.1 Introduction

The three procedures below allocate varying amounts of network resources for a single data item after each run of Dijkstra's algorithm, based on a cost function from Section 5. The performance of these procedures is shown in [ThT99] and Section 8.

The resource allocations performed by these procedures update the following information in the system after scheduling $Rq[i]$ to move, and before running Dijkstra's algorithm again: (1) the list of virtual links and their start and stop times, (2) the available memory capacity on any machines that data item $Rq[i]$ has been placed, (3) the list of machines on which $Rq[i]$ is available, and (4) the time at which $Rq[i]$ can be removed from any intermediate machines.

6.2 Partial Path Procedure

Each iteration of this procedure involves: (1) performing Dijkstra's algorithm for each data request individually; (2) for the valid next communication steps, determining the "cost" to transfer a data item to its successor in the shortest path; (3) picking the lowest cost data request and transferring that data item to the successor machine (making this machine an additional source of that data item); (4) updating system parameters to reflect resources used in (3); and (5) repeating (1) through (4) until there are no more satisfiable requests in the system. In some cases, Dijkstra's algorithm would not need to be executed each iteration for a particular data transfer, i.e., if the data transfer did not use resources needed for any future transfers. In this study, only one data item is scheduled before rerunning Dijkstra's algorithm (this applies for all three procedures). This simplified the implementation of the procedures without changing the performance of the resulting schedules. The execution time of the procedures is affected; however, minimizing this is not the main goal of the work.

'This section is based on [ThT99]. This material is needed background for the results presented in Sections 8 and 9, and is included here so that this report is self-contained.

This procedure will schedule the transfer for the single "most important" request that must be transferred next, based on a cost criterion. The procedure (first described in [TaT98]) is called the partial path procedure because only one successor machine in the path is scheduled at each iteration. If a data item is partially scheduled through the system and because of other scheduled transfers the requesting destination's deadline is no longer satisfied, the scheduled transfers remain in the system (the initial transfers were scheduled because the deadline could have been satisfied). Reasons the schedule for this now unsatisfiable request is not removed include: (1) in a dynamic situation, a change in the network could allow the request to be satisfied; and (2) removing the already scheduled transfers would require restarting the scheduling for all data requests because of conflicts that might have occurred.

6.3 N l Path/One Destination Procedure

The full path/one destination procedure uses a cost criterion to select a data request at an individual destination machine for resource allocation. The data item is then sent from its current location (machine $M[s]$ in each of the cost criteria) over as many virtual links as required to reach its destination machine (machine $M[j]$ for one value of j). For cost $C1$, the choice of j (i.e., which requesting destination should be satisfied) is trivial; $C1$ only takes into account a single requesting destination. All other cost criteria identify a set $Drq[i, \mathrm{r}]$ of destinations, and one destination $M[j]$ must be selected from that set to satisfy. For cost $C2$, the value of j chosen is the one satisfying the condition

$$\max_{j \in Drq[i,r]} Urgency[i, j]$$

from the equation describing C2. For cost C3, the value of j chosen is the one satisfying the condition

$$\min_{j \in Drq[i,r]} \frac{Efp[i, j]}{Urgency[i, j]}$$

from the equation describing $C3$. For costs $C4$, $C4links$, $C4size$, and $C4sizlnk$, the data item with minimum cost $Rq[i]$ is sent first to machine $M[r]$, and if no request was satisfied, the cost is applied a second time for the same data item $Rq[i]$, but setting the new $M[s]$ (data source machine) to the old $M[r]$ (the machine to which the data was just scheduled). The minimum cost is then taken over all values of r (possible next storage

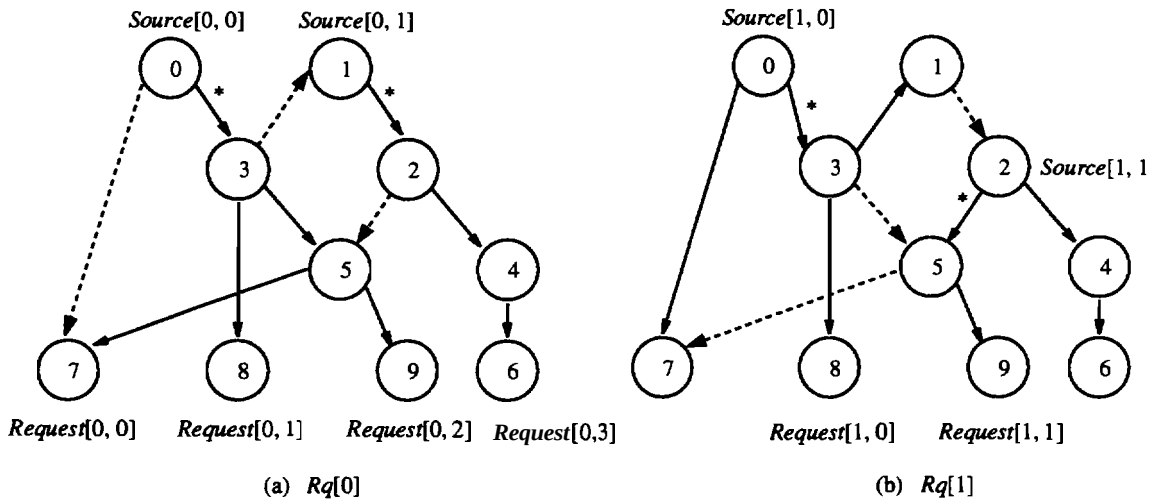

Figure 6.1: An example communication system that requests (a) $Rq[0]$ (corresponding to $\delta[0]$) and (b) $Rq[1]$ (corresponding to $\delta[1]$). $Source[k, j]$ denotes the jth initial source location of the kth data item $\delta[k]$. $Request[i, j]$ denotes the machine from which the jth request for data item $Rq[i]$ originates (in this example, $i = k$). Solid lines show shortest paths for a given data item to all nodes (even non-requesters), and dashed lines show unused links for a given data item. Asterisks denote next valid communication steps.

locations). The value of r with minimum cost determines the machine $M[r]$ that the data is sent to next. This process continues until the data item has reached one requesting destination M[j].

This produces a communication schedule using fewer executions of Dijkstra's algorithm than the partial path procedure. The behavior of the partial path procedure showed that if a data item $Rq[i]$ was selected for scheduling a transfer to its next intermediate location (a "hop"), in the following iteration, the same requested data item, $Rq[i]$, would typically be selected again to schedule its next hop. The full path/one destination procedure attempts to exploit this trend by selecting a requested data item with a cost criterion and scheduling all hops required for the data item to reach its lowest cost destination before executing Dijkstra's algorithm again.

Considering the example communication system in Figure 6.1(a), data item $Rq[0]$ would only be scheduled from $M[0]$ to $M[3]$ before executing Dijkstra's algorithm again in the partial path procedure. In the full path/one destination procedure, data item

$Rq[0]$ would be scheduled from $M[0]$ to $M[9]$ (a destination) before executing Dijkstra's algorithm again. This results in reducing the number of executions of Dijkstra's algorithm by three for this example. **A** savings proportional to the average length of a data item's path from a source to a destination is expected from this procedure. Considering again the communication system in Figure 6.1(a), if this procedure initially schedules the transfer of data item $Rq[0]$ from $M[0]$ to $M[9]$, $M[3]$ and $M[5]$ would become sources for $Rq[0]$. In the next iteration, $M[7]$ could receive $Rq[0]$ from $[M[5]$, and $M[8]$ could receive $\delta[0]$ from $M[3]$, without having to schedule a transfer from the original source, $M[0]$.

The partial path procedure may construct a partial path (of many links) that it later cannot complete (due to network or memory resources being consumed by other requested data items). However, until this is determined, the part of the path constructed may block the paths of the other requested data items, causing them to take less optimal paths or causing them to be deemed unsatisfiable. The full path/one destination procedure avoids this problem. An advantage the partial path approach does have over the full path/one destination approach is that it allows the link-by-link assignment of each virtual link and each machine's memory capacity to be made based on the relative values of the cost criteria for the data items that may want the resource.

6.4 N l Path/All Destinations Procedure

The full path/all destinations procedure resembles on the full path/one destination procedure but allocates more network resources after each run of Dijkstra's algorithm. This procedure satisfies all requests that would benefit from sending data item $Rq[i]$ from machine $M[s]$ to $M[r]$ as defined by one of the costs of Section 5. For all cost criteria except $C1$, this means that all of the destination machines in the set $Drq[i, \mathrm{r}]$ are satisfied by the resource allocation of this procedure. Cost $C1$ used in conjunction with this procedure only considers a single destination machine in its calculation, and as such would always satisfy only one requesting destination. Because this behavior is the same as using $C1$ with the full path/one destination procedure, criterion $C1$ is not used with the full path/all destination procedure.

For the example communication system in Figure 6.1(a), $Rq[0]$ is requested by machines $M[7]$, $M[8]$, and $M[9]$, and the shortest path for these three destinations all orig-

inate at machine $M[0]$ and pass through machine $M[3]$. The full path/all destinations procedure will schedule all paths for a single data item that share the next machine in the path as an intermediate machine. In Figure 6.1, the data item $Rq[0]$ would be scheduled for all three destinations (machines $M[7]$, $M[8]$, and $M[9]$) at the same time. By scheduling the path to multiple destinations, two fewer executions of Dijkstra's algorithm are required as compared to the full path/one destination procedure. A savings proportional to the average number of destinations for a data item whose shortest path intermediate machine set share a common machine is expected. This approach was considered because it was expected to generate results comparable to the full path/one destination procedure, but with a smaller procedure execution time.

7. Upper and Lower Bounds*

7.1 Introduction

Finding optimal solutions to data staging tasks with realistic parameter values are intractable problems. Therefore, it is currently impractical to directly compare the quality of the solutions found by the proposed heuristics with those found by exhaustive searches in which optimal answers can be obtained by enumerating all the possible schedules of communication steps. Also, to the best of the author's knowledge, there is no other work presented in the open literature that addresses the data staging problem and presents a heuristic for solving it (based on a similar underlying model). Thus, there is no other heuristic for solving the same problem with which to make a direct comparison of the heuristics presented in this document. To aid in the evaluation of these heuristics, two lower bounds and three upper bounds on the performance of the heuristics are provided.

7.2 Single Dijkstra Random

This lower bound shows the effect of running Dijkstra's algorithm only once for each requested data item, assuming it is the only requested item in the network. Then the paths through the network are scheduled for each data item, finishing $Rq[i]$ before $Rq[i+1]$ (where the ordering of the data items is arbitrary). Resources are allocated to data items one link at at time, and if a conflict arises, e.g., the link a transfer is attempting to schedule is no longer available, the data item stops progressing through the network and is not satisfied (but retains previously allocated resources). This approach is referred to as single Diikstra random because Dijkstra's algorithm is only executed once for each data item. This method is used to illustrate that executing Dijkstra's algorithm more than once, with updated communication system information, is advantageous.

'Subsections 7.2, 7.4, and 7.5 are based on [ThT99]. This material is needed background for the results presented in Sections 8 and 9, and is included here so that this report is self-contained.

7.3 N l Path Random Dijkstra

The lower bound called the full path random Dijkstra method does take into account which data requests are satisfiable when it allocates resources, allowing it to improve over the random Dijkstra method used in [ThT99]. It allocates enough resources in one scheduling step to take a data item from its current location all the way to one random satisfiable requesting destination before running Dijkstra's algorithm again. This method, based on the full path/one destination procedure, differs from the single Dijkstra random method of Subsection 7.2 in that (1) this method checks that a requesting destination is satisfiable before allocating any resources toward fulfilling it, and (2) Dijkstra's algorithm is run with updated communication system information after each scheduling step. If there is a much larger number of data requests in the system than could be satisfied, this method will execute much faster than the random Dijkstra method because it will not waste time running Dijkstra's algorithm to select and schedule requests that cannot be satisfied.

7.4 Upper Bound

This bound assumes that all requests in the system are satisfiable, and therefore represents the total weighted sum of the priorities of all requests in the system. This is an unrealistic (loose) bound because it does not take into account any network limitations that might prevent requests from being satisfied. It does, however, give an indication of the total value of the data being requested by the users of the system.

7.5 Possible Satisfy

If Dijkstra's algorithm was run to establish the satisfiability of each request as if it were the only request in the system, and the weighted priorities of each of the satisfiable requests were added together, the result would be the bound called possible satisfy. The reason that this is not equal to upper bound described above is that some requests cannot be satisfied due to lack of link bandwidth and/or machine storage, even when it is the only request in the system.

POSSIBLESATISFYBANDWIDTH(Rq, Nrq, Request, Priority, Worth, NetBandwidth)

1. invoke Dijkstra's algorithm for each request in Rq giving A_T
2. UsedBandwidth $\leftarrow 0$
3. PriSum $\leftarrow 0$
4. $max \leftarrow 1$
5. while UsedBandwidth $<$ NetBandwidth AND max > 0
6. $max \leftarrow 0$
7. for $i = 0$ upto $2\rho - 1$
8. for j $= 0$ upto $Nrq[i] - 1$
9. if $A_T[i, Request[i,j]] >$ Rf $t[i,j]$
10. if $(W[Priority[i,j]] * Worth[i,j] / |Rq[i]|) > max$
11. $i_{max} \leftarrow i$
12. $j_{max} \leftarrow j$
13. max $\leftarrow (W[Priority[i_{max}, j_{max}]] * Worth[i_{max}, j_{max}]) / |Rq[i]|$
14. if $max > 0$
15. $UsedBandwidth \leftarrow UsedBandw$idth $+ |Rq[i_{max}]|$
16. $PriSum \leftarrow PriSum +$ (W$[Priority[i_{max}, j_{max}]] * Worth[i_{max}, j_{max}]$)
17. remove destination j_{max} requesting $Rq[i_{max}]$ from the system
18. return PriSum

Figure **7.1:** The method used for the calculation of the possible satisfy bandwidth bound.

7.6 Possible Satisfy Bandwidth

The possible satisfy bandwidth bound is a tighter bound than the possible satisfy bound above. It considers satisfiable requests, and also the total amount of bandwidth available in the system, NetBandwidth. This value is calculated by adding together the number of bytes that could be transmitted over each virtual link in the system. The algorithm in Figure **7.1** shows how this bound is calculated. The loop of steps **7** through **13** select the request that is satisfiable and has the largest ratio of priority weight to data item size. Selecting the request that satisfies this condition guarantees that if a single link is used to satisfy this request, it will give the highest possible priority weight value per byte of network bandwidth used as compared to all requests remaining in the

system. Each time a request is found, steps **14** through **16** add its size in bytes to the bandwidth used in the system (this assumes that only one virtual link is needed to satisfy this request) and add its weighted priority to the weights of the other data items that have been selected. That particular request is then removed in step **17** so that a new request can be found. This upper bound is still unrealistic, however, because it does not take into account that more than one link may have to be used to satisfy a request, nor does it consider the time intervals that links are available, nor does it consider what machines have network bandwidth available between them.

8. Extended Simulation Study

8.1 Introduction

After the simulation study of [ThT99] was completed, a new study was designed to examine the effects of varying some other parameters within the system. In particular, this new study introduces three new cost criteria and two new bounds, and it varies additional simulation parameters, including eight network loadings, three average numbers of links used to get from a source machine to a destination machine, and five priority weighting schemes.

The results of [ThT99] indicated that C4 was the best-performing cost criterion. This led to the development of cost criteria $C4size$, $C4links$, and $C4sizlnk$, described in Section 5, for the new study. Because of the previous performance of the full path/one destination procedure, it was implemented for the new study with all seven cost criteria described in Section 5. For comparison, the other two procedures in Section 6 (partial path and full path/all destinations) were also implemented for the new study with cost C4, for a total of nine heuristics. Eight E-U ratios were tried for the two pairings using costs $C4size$ and $C4sizlnk$, and nine E-U ratios were tried for the other seven pairings using costs $C1$, C2, C3, $C4$, and $C4links$. Nine were needed in the latter case in order to determine the best E-U ratio.

In the previous study, all requests averaged traversing approximately 1.5 communication links (a communication link traversal count) from an initial source machine to a requesting destination machine. It was decided that the requests would be generated in a manner allowing this parameter to be controlled and varied with three different values in the new study. Another parameter concerning the data requests was the number of requests being made versus the number of requests that the network could possibly fulfill. Eight different "network loads" were decided upon for the new simulation study, in combination with the three communication link traversal counts, for a total of 24 different

Table 8.1
Network parameters used for the generation of test cases.

parameter	minimum value	maximum value
number machines	14	16
number sources per data item	1	3
number destinations per data item	1	5
source available time	1 sec	3600 sec
destination deadline delay	900 sec	3600 sec
data item size	10 kBytes	100 MBytes
machine storage	10 MBytes	20 GBytes
machine outbound link degree	1	4
link bandwidth	10 kBits/sec	1.5 MBits/sec

data request scenarios.

For this study, it was decided that a six-level priority scheme would be used in place of the three-level method used in the previous study. This was intended to better reflect the priority classes present in a military environment. In addition, the weighting of these priority levels was changed to a system where the weight of each priority level was a fixed multiple of the weight of the priority level immediately below it. Five different values for this multiple were used for this study, and each was evaluated with each of the 24 data request scenarios above, resulting in 120 testing scenarios for evaluation by the 79 heuristic/E-U ratio combinations.

As in the previous study, 40 individual test cases (each with a unique network configuration and set of data requests) were generated for each testing scenario, because a single case cannot reflect the range of possible data requests and network configurations. This resulted in the 379,200 simulation runs described in this section.

8.2 Generation of Test Cases

The network parameters used to create data sets for this simulation study are summarized in Table 8.1. Actual values are generated randomly with uniform probability

between (and including) the minimum and the maximum values shown in the table. These parameter values are intended to be representative of a subset of a BADD-like environment.

The "number machines" parameter refers to the number m of machines in the network. "Number sources per data item" is the number of source machines $N\delta[i]$ (where $0 \leq i <$ n) that initially hold data item $\delta[i]$, and is generated independently for each value of i. "Number destinations per data item" is the number of destination machines $Nrq[j]$ (where $0 \leq$ j $<$ *p)* that have requested a copy of data item $Rq[j]$ (only the higher quality data items are considered in this study). This value is also generated independently for each value of j. Each requesting destination for each data item also has a priority class α (where $0 \leq \alpha \leq 5$), where class 0 is generated with a 50% probability, class 1 with 25%, class 2 with 12%, class 3 with 7%, class 4 with 4%, and class 5 is generated with a 2% probability. These percentages were selected to reflect the fact that in a BADD-like environment, there would likely only be a small number of data requests in the highest priority class, and a large number of data requests at the lowest priority class.

The "source available time" in Table 8.1 corresponds to $\delta st[i, \text{k}]$ (where $0 \leq$ k $< N\delta[i]$), the time at which data item $\delta[i]$ is available at its kth source machine. The time is given as an offset in seconds from the beginning of the time interval being simulated. For these simulations all available times for a given data item are equal (i.e., $\delta st[i, 0] = \delta st[i, 1] = \ldots = \delta st[i, \text{k} - 1]$). The "destination deadline delay" refers to the number of seconds between the time that the data item is available at its source machine(s) and the time that it is needed at a destination machine (the deadline). This delay is generated independently for each destination machine of a given data item. Because data items may become available up to one hour after the beginning of a simulation interval, and a data item may have up to one additional hour before its deadline at a destination, the total simulation interval is two hours.

The time duration parameter for garbage collection at intermediate machines γ, was set to six minutes. This means that the removal time δrt for a data item at a machine that is not an original source machine nor a requesting destination machine is six minutes after the latest deadline Rft for that same data item at any requesting destination. Source

machines and final destination machines for data items hold those data items for the remainder of the simulation period (δrt is ∞).

The "data item size" $|\delta[i]|$ (where $0 \leq i < n$) mentioned in Table 8.1 is generated for each data item and affects the amount of storage required to hold a data item as well as the amount of time required to transmit a data item on a virtual link. "Machine storage" $Cap[j](t_0)$ (where $0 \leq j < m$ and t_0 is the beginning of the simulation interval) is the amount of unused storage space on a machine in the network. The "machine outbound link degree" refers to the number of unidirectional outbound communication links that a machine in the network has. Link generation is done for each machine in the network, and additionally ensures that a link must terminate at a different machine than it originated from, and no more than two links originating from one machine can terminate at the same destination machine. The bandwidth of each link is in the interval defined by the "link bandwidth" of Table 8.1.

Unidirectional communication links are intended to represent links that may only be available during certain periods of the day, such as satellite links. Communication link availability is calculated for a 24 hour period in the following manner. For each communication link, the percentage of the day that the link is to be available is randomly chosen from the set {50%, 60%, 70%, 80%, 90%, 100%). If the link is available for 100% of the day, a single virtual link is initially used to represent the communication link for the entire 24 hour period. However, if the link is chosen to be available for less than 100% of the day, an availability duration is randomly chosen from the set {30 minutes, one hour, two hours, four hours}. This duration represents the length of time that the communication link will initially be continuously available each time it becomes active. Each period of availability will then be represented by a single virtual link with a starting and ending time.

The number of virtual links used is determined by the ceiling of dividing the amount of time the link is available (percentage available * 24 hours) by the availability duration chosen. The starting time of the earliest virtual link (as an offset from the beginning of the day) is randomly chosen between 0 and one tenth of the total unavailable time of the communication link. Unavailable times are generated in a similar manner between

each of the remaining virtual links until the appropriate number of virtual links has been allocated. If all of the unavailable time is allocated before the last virtual link, then there is no unavailable time allocated between remaining virtual links.

The machines and unidirectional communication links of a network are generated with the preceding parameters, and then tested to make sure that there is a path from each machine to all other machines via some set of communication links. An adjacency matrix A_{links} is calculated where each entry corresponds to the minimum number of communication links traversed (communication link traversal count) in a path from one machine to another machine. For this calculation, all links are assumed to have infinite bandwidth and be available for the entire simulation duration. This means that each entry in the adjacency matrix is a lower bound on the number of communication links that would be used to send a data item from one machine to another. After this matrix is created, each data item is created, along with a set of source machines to hold the item initially, and a set of destination machines to request the data item.

The behavior of all three procedures depends on the communication link traversal count. The expected behavior is that for a larger average count, the full path procedures should execute faster than the partial path procedure because they allocate all resources required to satisfy a request at once, whereas the partial path procedure only allocates one virtual link at a time. As the count increases, it also becomes more possible for the partial path procedure to "strand" a data item at an intermediate storage location resulting in a poorer overall schedule (i.e., the procedure would allocate enough resources for a particular data request to move it toward its requesting destination, but would not allocate enough resources to cause the data request to be satisfied).

For these reasons, the average communication link traversal count was varied with three arbitrary values: 1.5, 2.5, and 3.5. These values are shown in Table 8.2, each with four experimentally determined probabilities. As an example of what these percentages mean, consider the bottom line of the table with a resulting average count of 3.5. In order to generate a network with satisfiable requests that have an average communication link traversal count of 3.5, there must be a 1% chance of generating a data request that is one communication link away from a source machine, a 3% chance of generating a data

Table 8.2
Experimentally determined probabilities of communication link traversal counts from a source machine to a requesting destination machine and their resulting average link counts.

probability of communication link traversal count				resulting average communication link traversal count for satisfiable requests
1	2	3	4	
80%	19%	1%	0%	1.5
17%	38%	45%	0%	2.5
1%	3%	81%	15%	3.5

request with a communication link traversal count of two, an 81% chance of making a request with a count of three, and a 15% chance of creating a request with a count of four.

The process of generating a single data item and its associated source machine(s) and requesting destination machine(s) is now described. First, the number of source machines m_s and the number of destination machines m_d are chosen with uniform distribution (see Table 8.1 again for ranges). For each destination machine, a desired communication link traversal count $L_{cnt}(i)$ (where $0 \leq i < m_d$) is generated according to the probabilities listed in Table 8.2, depending on the resulting average communication link count wanted. For example, if the desired average count wanted was 3.5, $L_{cnt}(i)$ for all values of i would be 3 with probability 81%. These probabilities were determined experimentally.

Then, all possible sets of source machine combinations of size m_s are enumerated in a list where an element (a set of source machines of size m_s) is identified as $m_{sets}(j)$ (where $0 \leq j < \binom{m}{m_s}$). Using the adjacency matrix A_{links}, for each element $m_{sets}(j)$, a list of communication link traversal counts $Lc_{min}(j)$ is calculated. The kth (where $0 \leq k < m - m_s$) element in the list $Lc_{min}(j)$ is the minimum communication link traversal count from any of the source machines in the set $m_{sets}(j)$ to the kth destination machine.

Now, for each value of i such that $0 \leq i < m_d$, consider the count in $L_{cnt}(i)$. Then, for

every list $Lc_{min}(j)$, try to remove an element in that list which is equal to the link count $L_{cnt}(i)$. This removal corresponds to reserving a potential destination that is a minimum of $L_{cnt}(i)$ communication links away from all of the m, source machines in $m_{sets}(j)$. If there is a list Lc_{min}(j) that does not contain a value equal to the link count $L_{cnt}(i)$, delete that list $Lc_{min}(j)$ and the corresponding set of source machines $m_{sets}(j)$

Remaining in m_{sets} is now the source machine combinations that have appropriate communication link traversal counts to potential destination machines in the network. If all source machine combinations have been removed from m_{sets}, the data item is regenerated from the beginning. Otherwise, one of the remaining source machine combinations is randomly selected to be the set of source machines for the current data item. Destination machines can then be selected 'that fit each of the desired communication link traversal counts in $L_{cnt}(i)$.

For this simulation study, the number of data items generated for a network was 700 times the number of machines in the network. After all items were generated, Dijkstra's algorithm was run once for each item, establishing the individual satisfiability of each data item at each requesting destination along with a path of communication links used to reach each destination. The average number of communication links traversed from a source machine to a destination machine for all of the satisfiable requests is the "resulting average communication link traversal count" shown in Table 8.2. As indicated above, three different average link counts were generated, and for each count, 40 different networks and associated data requests were created with the method given above, resulting in a total of 120 networks with associated data requests.

Now consider in the network all data requests that are determined to be satisfiable individually according the first execution of Dijkstra's algorithm. When considering each of these requests as if it were the only data request in the system, the resulting virtual link path from Dijkstra's algorithm and other known information can be used to calculate the bytes of bandwidth needed for each request. Then these bandwidths can be summed to give a value representing the total number of bytes of data bandwidth being requested in the system. Call this value *ReqBandwidth*. Recall now the value *NetBandwidth* calculated by summing together the total number of bytes that could be

transmitted on each of the virtual links within the network during the simulation period. An oversubscription rate can then be defined as $ReqBandwidth/NetBandwidth$. If this term is larger than 1, the network can clearly not satisfy all requests due to bandwidth limitations. If the term is less than 1, bandwidth may not exist between the correct machines or may not be available during the required time to satisfy all requests.

In order to examine system performance under various request loads, it was decided to consider networks with the following oversubscription rates: 25.0, 12.5, 6.2, 3.1, 1.6, 0.8, 0.4, and 0.2. These desired data sets were created by starting with one of the networks and its associated set of data requests, and removing random data requests until the desired oversubscription rate was achieved. This did not significantly affect the average communication link traversal counts. It resulted in data sets consisting of the same network with eight different oversubscription rates, for each of the 120 networks.

When applying the heuristics to these test cases, a variety of E-U ratios were used. For simulations run using the full path/one destination procedure with $C4size$ and $C4sizlnk$, the E-U ratios used were 10^{inf}, 10^{9}, 10^{8}, 10^{7}, 10^{6}, 10^{5}, 10^{4}, and 10^{-inf}. The values of 10^{inf} and 10^{-inf} represent considering only the priority term (the term weighted by W_E), and only the urgency term (the term weighted by W_U), respectively. For simulations run using the partial path procedure with C4, the full path/all destinations procedure with C4, and the full path/one destination procedure with $C1$, C2, C4, and $C4links$, the E U ratios used were 10^{inf}, 10^{6}, 10^{5}, 10^{4}, 10^{3}, 10^{2}, 10^{1}, 10^{0}, and 10^{-inf}. Recall that C3, which was implemented in this study with the full path/one destination procedure, does not have a W_E or W_U term.

The last parameter that was varied in this simulation study was the relative weight of one level of priority request compared to another. With the six priority levels of data requests, the approach simulated was to make the weight of a priority level a (where $0 \leq a \leq 5$) data request be ω^{α} (i.e., $W[\alpha] = \omega^{\alpha}$) for some fixed value of w. The values of ω simulated were 1, 2, 4, 8, and 16, and this was done for each of the networks and loadings mentioned above. The results of the simulations using these parameters are now presented.

Table 8.3
Labels for heuristics and bounds used in the graphs of Subsection 8.3.

implementation combination	label used
partial path procedure with C4	partial–C4
full path/one destination procedure with C1	full–one–C1
full path/one destination procedure with C2	full–one–C2
full path/one destination procedure with C3	full–one–C3
full path/one destination procedure with C4	full–one–C4
full path/one destination procedure with C4*links*	full_one_C4links
full path/one destination procedure with C4*size*	full_one_C4size
full path/one destination procedure with C4*sizlnk*	full–one–C4sizlnk
full path/all destinations procedure with C4	full–all–C4
upper bound	upper–bound
possible satisfy	possible–satisfy
possible satisfy bandwidth	possible–satisfy–bandwidth
full path random Dijkstra	full–rand–Dijkstra
single Dijkstra random	single–Dijkstra–random

8.3 Evaluation of Simulations

Heuristic and bound labels used in the graphs at the end of this subsection are summarized in Table 8.3. In [ThT99], graphs were shown with the performance of most of these heuristics versus E U ratio. The three new costs taking into account data item size and the number of communication links traversed from a source to a destination are shown in Figure 8.1 (all Section 8 figures are at the end of the section). The peak performance of the costs taking data item size into account are further to the right (signifying higher E U ratios) in the graph because those costs divide the effective priority term by the data item size. The points inf and – inf are the two extremes, where inf only considers the effective priority term, and – inf only considers the urgency term. The same set of E U ratios was not used for all three cost criteria (as detailed in Subsection 8.2); for

this graph, the 10^{-inf} E-U ratio data points for full−one−C4size and full−one_C4sizlnk were duplicated for their 10^0, 10^1, 10^2, and 10^3 E-U ratio data points, and the 10^{inf} E-U ratio data point for full_one_C4links was duplicated for its 10^7, 10^8, and 10^9 E-U ratio data points.

The graphs of Figures 8.2 through 8.31 show the effects of varying the oversubscription rate, average link traversal count, and the priority weighting parameter, w. The graphs of Figures 8.2 through 8.16 show all of the upper bounds from Section 7, as well as the full path random Dijkstra (Subsection 7.3) and single Dijkstra random (Subsection 7.2) lower bounds. The random Dijkstra (used in [ThT99]) lower bound is not included in these simulation results because of the large execution time required for it to run in the heavily loaded test cases. In addition, Figures 8.2 through 8.16 show data points labeled best−hc and worst_hc, corresponding to the best data points generated by any of the heuristics (procedure/cost criterion pairs), and the worst data points generated by any of the heuristics, respectively. The data points for the heuristics used correspond to the best E-U ratio for each testing scenario. The best E U ratio was only likely to be 10^{inf} or 10^{-inf} in tests where $w = 1$; the best performance in other tests was almost always a combination of both the priority and urgency terms. The values for the normalized vertical axis in all of these graphs is computed as follows. For each test case, the sum of the satisfied requests' weighted priorities for a given heuristic or bound is divided by the sum of satisfied requests' weighted priorities given by the best E-U ratio for full−one−C4. This normalized sum is then averaged over the 40 network test cases to give the final value for each data point.

Figures **8.2** through 8.4 show the relative performance of the bounds and best and worst heuristics in a system where all data items have the same priority ($w = 1$). Notice in these figures that possible−satisfy−bandwidth levels off soon after the request over-subscription rate exceeds 1. This indicates the point at which the network can no longer satisfy more requests, although its gradual increase shows that as more smaller data items (which have a larger ratio of value per byte) are added to the system, they could be satisfied in place of larger data items. The difference between possible−satisfy and upper−bound is all of the data requests that are made that cannot be satisfied because

no virtual link exists between the appropriate source and destination machines.

As can be seen as the communication link traversal count increases from 1.5 to 3.5 in Figures 8.2 through 8.4, the difference between upper−bound and possible−satisfy increases. This is because a longer average communication link traversal count will generally require data items to use more virtual links to get from a source to a destination, thus making it harder to meet deadline times. It should also be noted that in these figures, the performance of the heuristics decreases with respect to the upper bounds as the link traversal count increases. This is explained by the fact that the upper bounds do not have requests competing for bandwidth (i.e., each request is treated as if it is alone in the system), while the heuristics run out of bandwidth faster if the requests that they allocate resources for have higher link traversal counts. This trend is seen across all of Figures 8.2 through 8.16 as well.

Figures 8.2 through 8.4 also show that the performance of the worst heuristic is sometimes worse than the performance of the full path random Dijkstra lower bound. It should be noted that these test cases set $w = 1$, and hence the effective priority of all requests in the system is identical. There are two reasons for this poor relative performance of worst−hc. First, the effective priority term is unable to be utilized well by the cost criteria because all effective priorities are set to one. Second, the fact that full_rand_Dijkstra pays no attention to the effective priority of requested data items is not a hindrance in this case where all data items have equal priority.

Also shown in Figures 8.2 through 8.16 is the trend that as ω increases, the performance of the heuristics as well as the possible−satisfy−bandwidth bound become closer to the possible−satisfy and upper−bound bounds. This is because as ω increases, the sum of the weights of the highest priority class requests represent a larger percentage of the total sum of priority weights of requests. In other words, just satisfying the requests of priority class 5 results in satisfying a significant percentage of the sum of all requests' priority weights.

The relative performance of the heuristics are shown in Figures 8.17 through 8.31. The first three graphs, Figures 8.17 through 8.19, again show performance in a system where all data requests have the same priority ($w = 1$). In these three figures, the heuristics

divide themselves roughly into three groups of similar performance. The group with the best performance, consisting of full—one—C4size and full_one_C4sizlnk, contains the two methods that consider data item size. Because all effective priorities are the same in this system, data item size becomes an important way to distinguish between a good and a poor data item choice for resource allocation for the more heavily loaded test cases. The reason that data item size does not help for the more lightly loaded test cases is because of a fragmentation problem. The costs considering data item size will tend to allocate resources for all of the smaller data items first, resulting in many small time intervals of link bandwidth being allocated initially. In these lightly loaded cases, the remainder of the link bandwidth must be used by larger data items, but no continuously available links exist for a long enough period of time for these larger data items to use. In the more heavily loaded network cases, there are enough smaller data items available to make use of all of the network bandwidth without sending any of the larger data items. The resulting trend, shown in varying degrees in all of Figures 8.17 through 8.31, is that the costs incorporating data item size have a relative decrease in performance for lightly oversubscribed networks, followed by a relative increase in performance for the heavily oversubscribed networks.

The group with the worst performance in Figures 8.17 through 8.19 consists of full_one_C1 and full—one—C3. The full—one—C1 method is disadvantaged by the fact that it only considers the benefit of one data request at a time, whereas the heuristics of the group in the middle all consider multiple data requests that would collectively benefit from a resource allocation. Finally, the full—one_C3 method has a strong tendency to select very urgent requests. The urgency of a request is related to its size in that a larger data item will tend to arrive later at its destination because of the amount of bandwidth needed to transfer it. Therefore, this method will end up selecting the largest data items first, resulting in a lower ratio of weighted priority per byte for its satisfied requests. Note that $w = 1$ is a special case for full_one_C1, because for the graphs of 8.20 through 8.31, its performance remains close to that of the best heuristic.

In the remaining graphs (Figures 8.20 through 8-31), a number of trends can be seen. There is a general overall trend that as w increases (and other factors are fixed), the

performance of all heuristics is closer to each other. This, as in Figures 8.2 through 8.16, is because more of the total sum of priority weights of requests in the system is contributed by a few highest priority requests.

The full−one−C3 method performs consistently poorly for heavily oversubscribed networks. Its performance in the simulation studies of [ThT99] indicated that it would not likely perform well, so this was expected. It is interesting to note, however that as w was increased, the relative performance of full−one−C3 increased as well. This suggests that the problem with cost $C3$ is indeed due to allowing the urgency factor to dominate the cost equation, because as the priority weight is increased, it begins to perform well. This is especially true for the lower oversubscription rates, as seen in Figures 8.26, 8.27, 8.29, and 8.30.

The two methods that take into account data item size, full−one−C4size and full_one_C4sizlnk, are shown to perform well with high link traversal counts and very high oversubscription rates, as seen in Figures 8.25, 8.28, and 8.31. This indicates that data item size, for the values and distribution tested here, is not an important cost factor unless the system is very heavily oversubscribed. As mentioned previously,a reason that size may be a hindrance in more lightly oversubscribed systems is network fragmentation. If small data items are selected for resource allocation first, they will reserve small time intervals of virtual link bandwidth. Later, when larger data items are considered, there will not be enough continuous bandwidth available on any virtual links for transferring these data items. Only if the oversubscription rate is very high are there enough small data items available to utilize the remaining link bandwidth. For w = 1, in general, full−one−C4size and full_one_C4sizlnk do not perform as well as full−one−C4 because of this fragmentation problem.

In Figures 8.20 through 8.31, full_one_C1 performs well except for the highest oversubscription rate test cases. Because cost $C1$ only considers the benefit of moving data to satisfy a single request, this suggests that in very highly oversubscribed networks, it helps to consider multiple requesting destinations that would collectively benefit from a data transfer.

The full−one−C2 method appears to suffer from its choice of destination machines;

specifically, it chooses the most urgent request from a set of requesting destinations that would benefit from a common data transfer. As system oversubscription rates increase, its relative performance decreases in all of Figures 8.20 through 8.31. The method full_one_C4links, however, performed very comparably to full_one_C4 in all tests. There was no situation indicated by these simulations where full_one_C4links should be chosen over full_one_C4, or vice versa. The partial−C4 method was also shown to perform comparably to the full−one−C4 method in all cases.

The full−all−C4 method is shown to perform well for small average link traversal counts, but as the link traversal count increases, Figures 8.20 through 8.31 show a clear decrease in performance. This is due to the full path/all destinations procedure allocating resources for more than one destination simultaneously, where some requesting destinations may have very low priority.

Execution times for all bounds and heuristics are shown in Figure 8.32, and the four heuristics that tended to perform the best for $\omega > 1$ (full_one_C4links, full_one_C1, full−one−C4, and partial−C4) are shown by themselves in Figure 8.33. The full−one−C4 method does have an advantage in execution time over partial_C4, as shown in Figure 8.33. Figure 8.32 shows an example of the trends of execution times for each algorithm; execution times for other oversubscription rates do vary (larger execution times for larger oversubscription rates), but the trends remain the same. The general downward trend of all algorithms as link traversal count increases is due to the fact that requests that are deemed to be unsatisfiable by a given run of Dijkstra's algorithm do not have Dijkstra's algorithm applied to them again. With higher link traversal counts, network resources are used up more quickly, resulting in fewer remaining satisfiable requests, thus shortening execution time.

Figures 8.34, 8.35, and 8.36 show 95% confidence intervals (i.e., given the calculated sample mean over the 40 test cases, the probability that the true mean is in the interval shown is 0.95; see [Cas93]) corresponding to the graphs of Figures 8.23, 8.24, and 8.25. For a given path length and a given heuristic, the confidence interval is similar for any value of w or for any oversubscription rate. These sample graphs show the overall trend that as the path length increases, the confidence interval increases. The largest confidence

intervals calculated for any data points were always less than ±3%. The approximate worst case intervals were ±3% for link traversal counts of 3.5, ±2% for link traversal counts of 2.5, and ±1% for link traversal counts of 1.5. A majority of the intervals calculated were less than ±1%. Note that full−one−C4 is not shown in these graphs because it is being normalized to itself and hence always has a normalized performance value of 1.

Tables 8.4 and 8.5 show the average number of requests satisfied at each priority level by full−one−C4 as compared to a simple algorithm that schedules all requests of a higher priority level before any requests of a lower priority level. In particular, this algorithm was full_one_C1 with an E U ratio of 10^{inf}. For $w > 1$ in these tables, more requests in the top three priority levels are being satisfied by full−one−C4 (which obeys the relative importances assigned to each of the priority levels set by the policy maker) than the level by level method (which ignores these policy requirements). The number of satisfied requests at the top priority level remains comparable for full−one−C4 and $\omega > 1$ because there are so few requests at that level that all are able to be satisfied. This is indicated by the fact that the level by level method cannot satisfy any more of the top priority requests. For example, even though the level by level method schedules all priority level 5 requests as if they were the only requests in the system, the total number scheduled does not exceed the results of full−one−C4 (for $w > 1$). This shows that full−one−C4 using urgency in addition to effective priority, is better than full_one_C1 without urgency. Furthermore, full−one−C4 results in a higher sum of weighted priorities of satisfied requests than the level by level method in almost all cases considered in Tables 8.4 and 8.5.

In summary, a class of heuristics that compare well to upper and lower bounds has been developed and analyzed. Many heuristics perform within a few percentage points of each other, and this is why we consider the execution times of the different approaches. Furthermore, while in general several heuristics perform comparably, if a system is known to have a particular operating environment (e.g., w value, oversubscription rate), there may be a preference for one pair over another. Future work will investigate confidence intervals for some of the data points generated by test cases in this section.

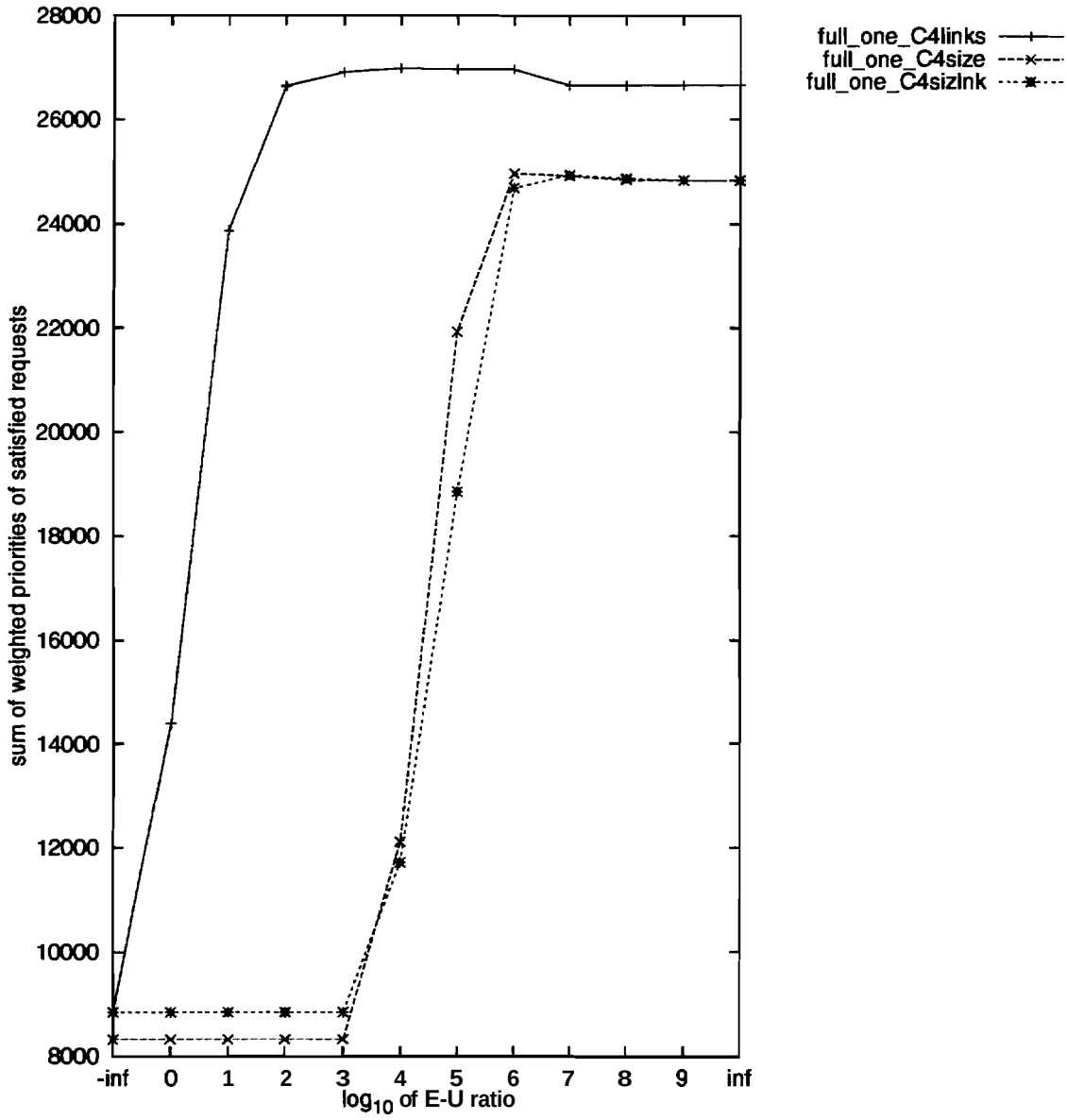

Figure 8.1: Sample graph of the effect of varying the E U ratio with the three new cost criteria. The data sets used had an average link traversal count of 2.5, a request oversubscription rate of 3.1, and an w value of 4. The $10^{-\inf}$ E-U ratio data points for full—one_C4size and full—one—C4sizlnk were duplicated for their 10^0, 10^1, 10^2, and 10^3 E-U ratio data points, and the $10^{\inf}$ E-U ratio data point for full—one—C4links was duplicated for its 10^7, 10^8, and 10^9 E-U ratio data points.

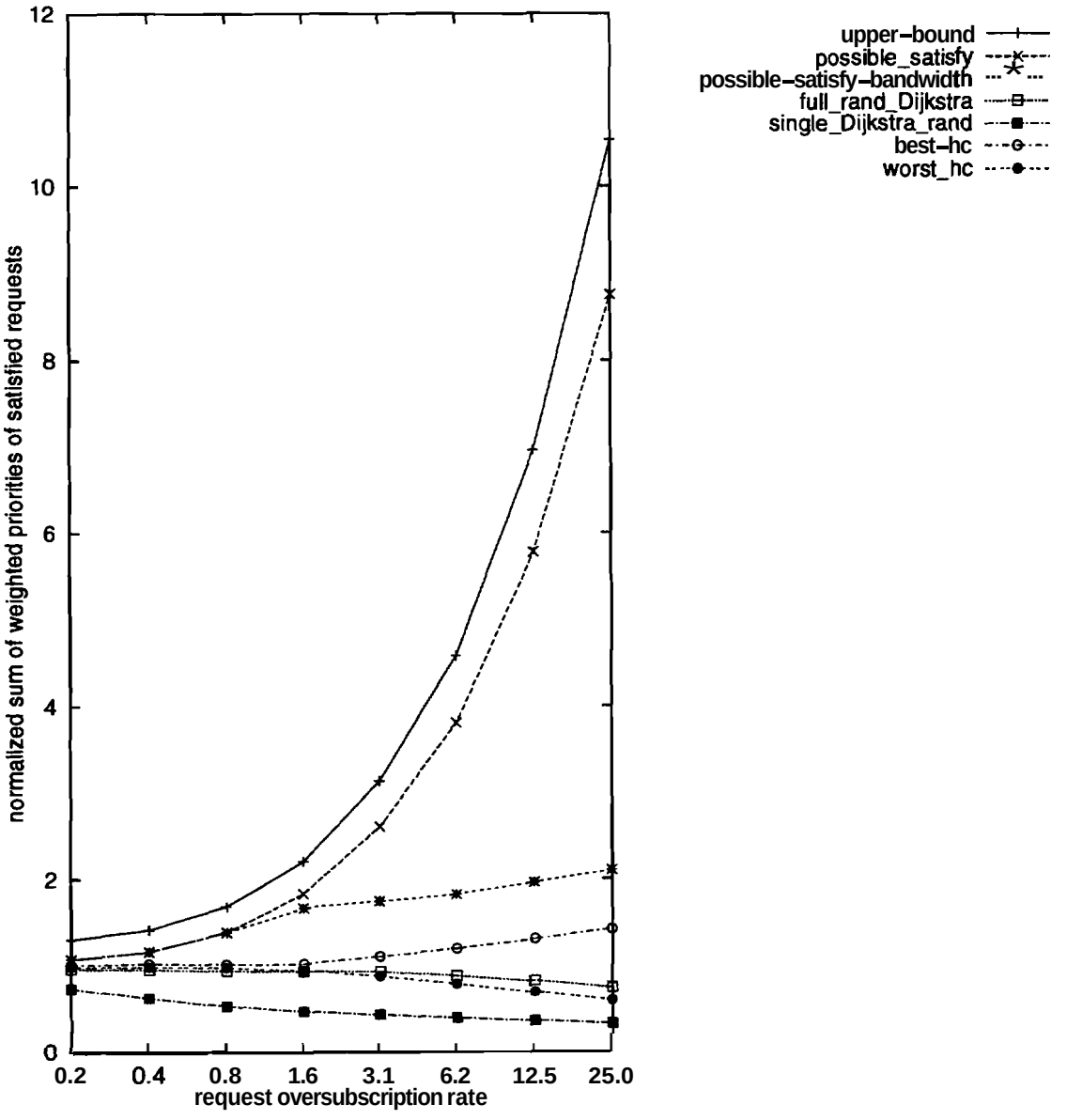

Figure 8.2: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full-one-C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 1.5 **and an** w **value of 1.**

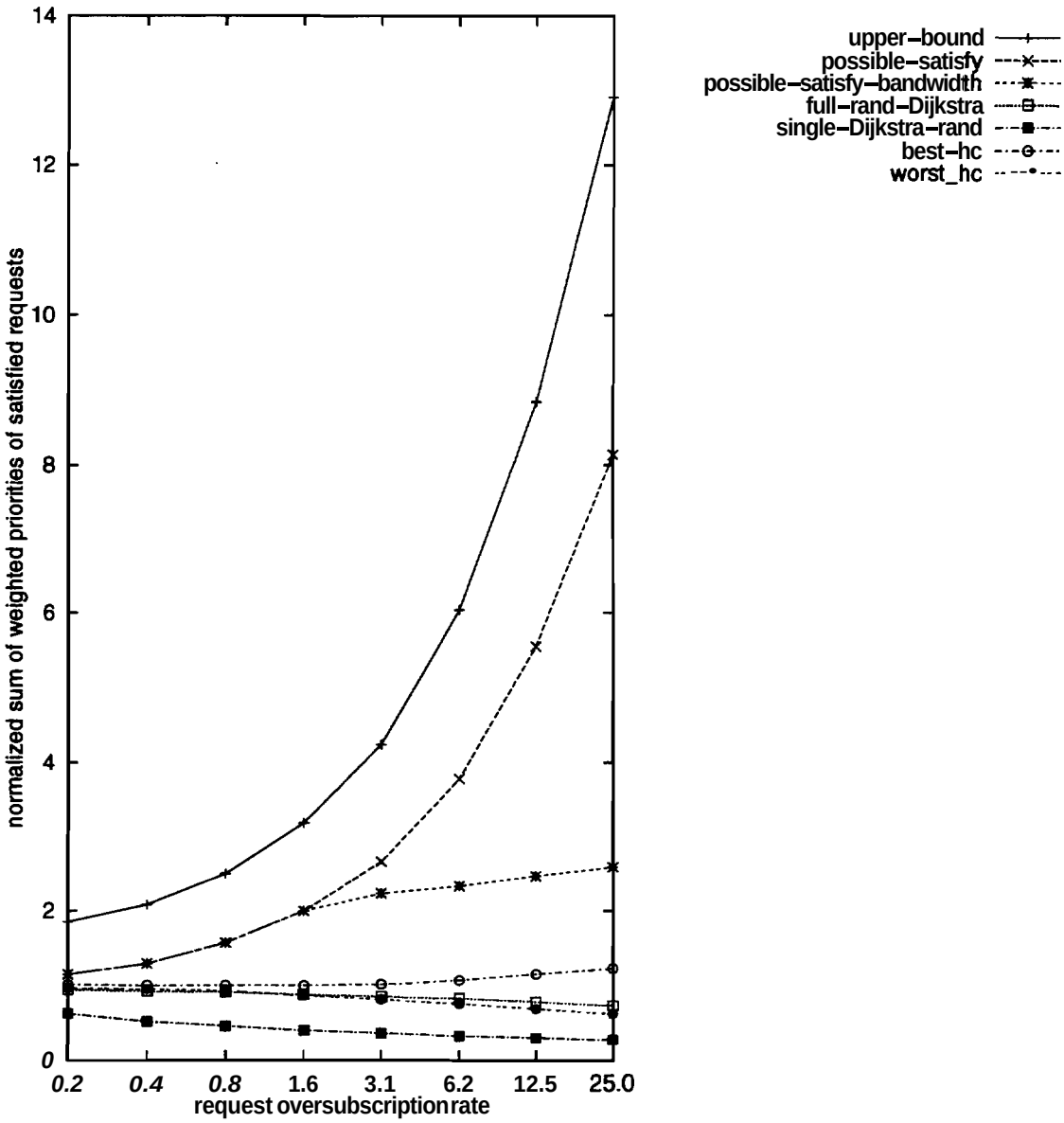

Figure 8.3: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an w value of 1.

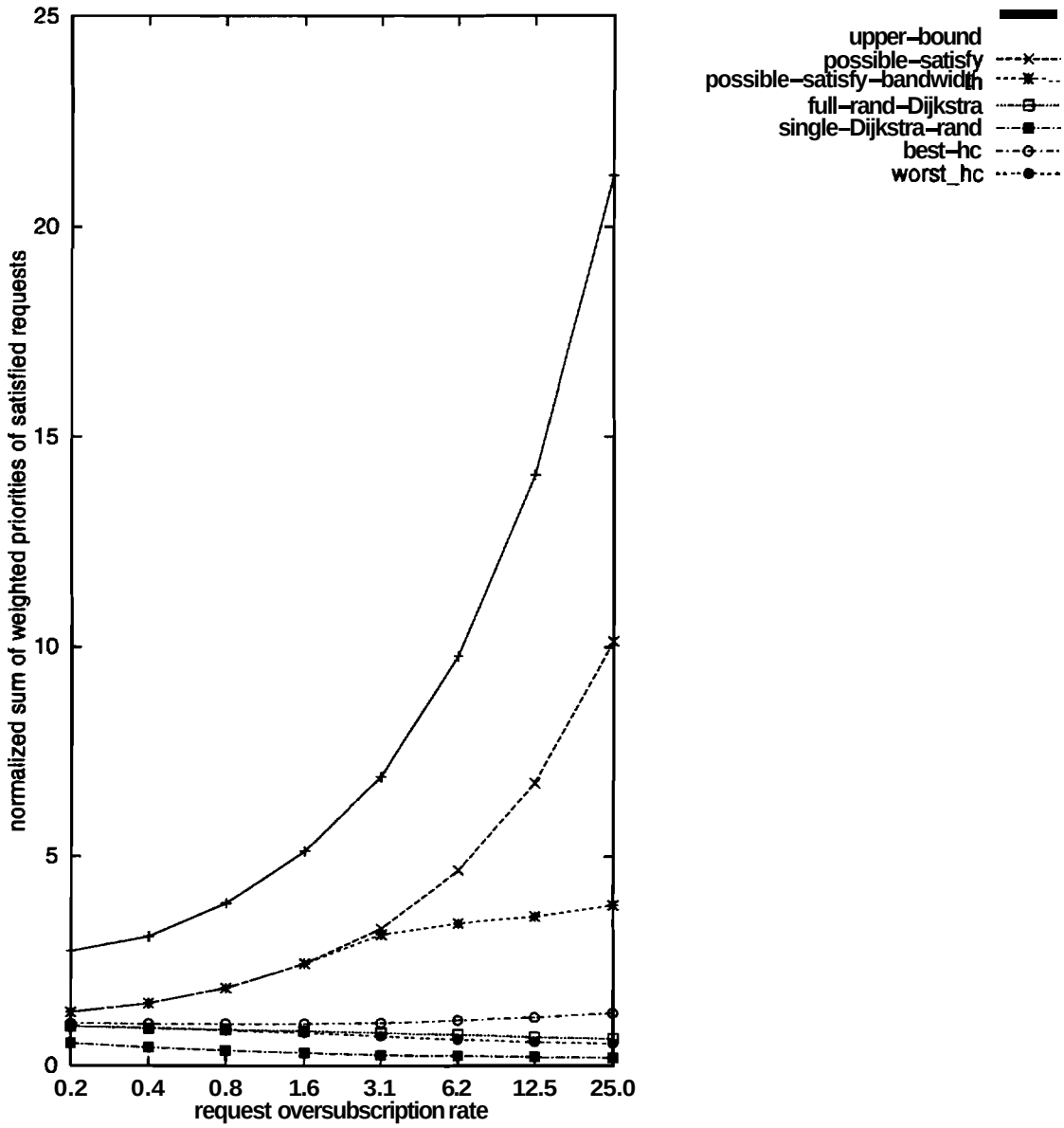

Figure 8.4: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an w value of 1.

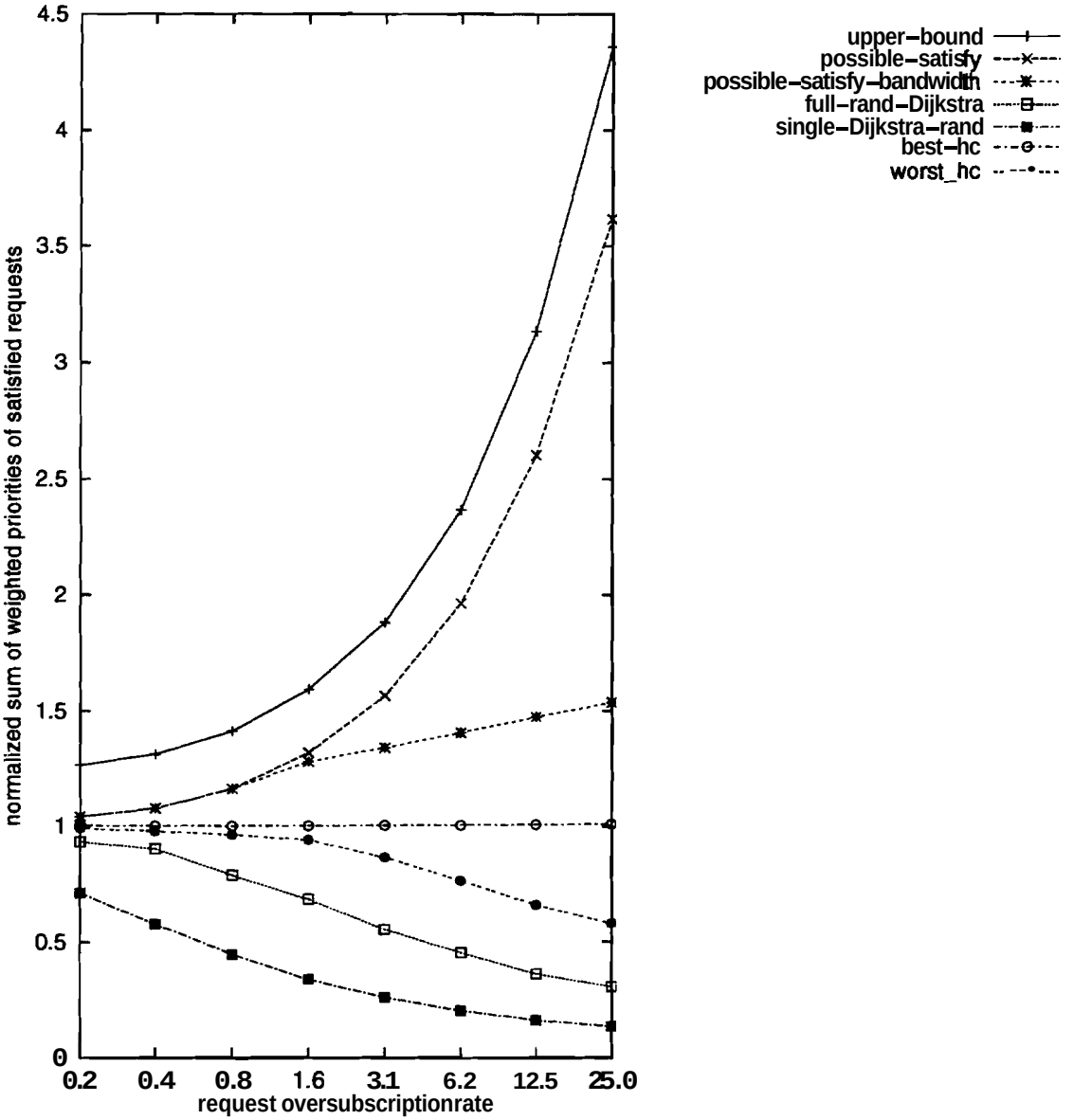

Figure 8.5: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an w value of 2.

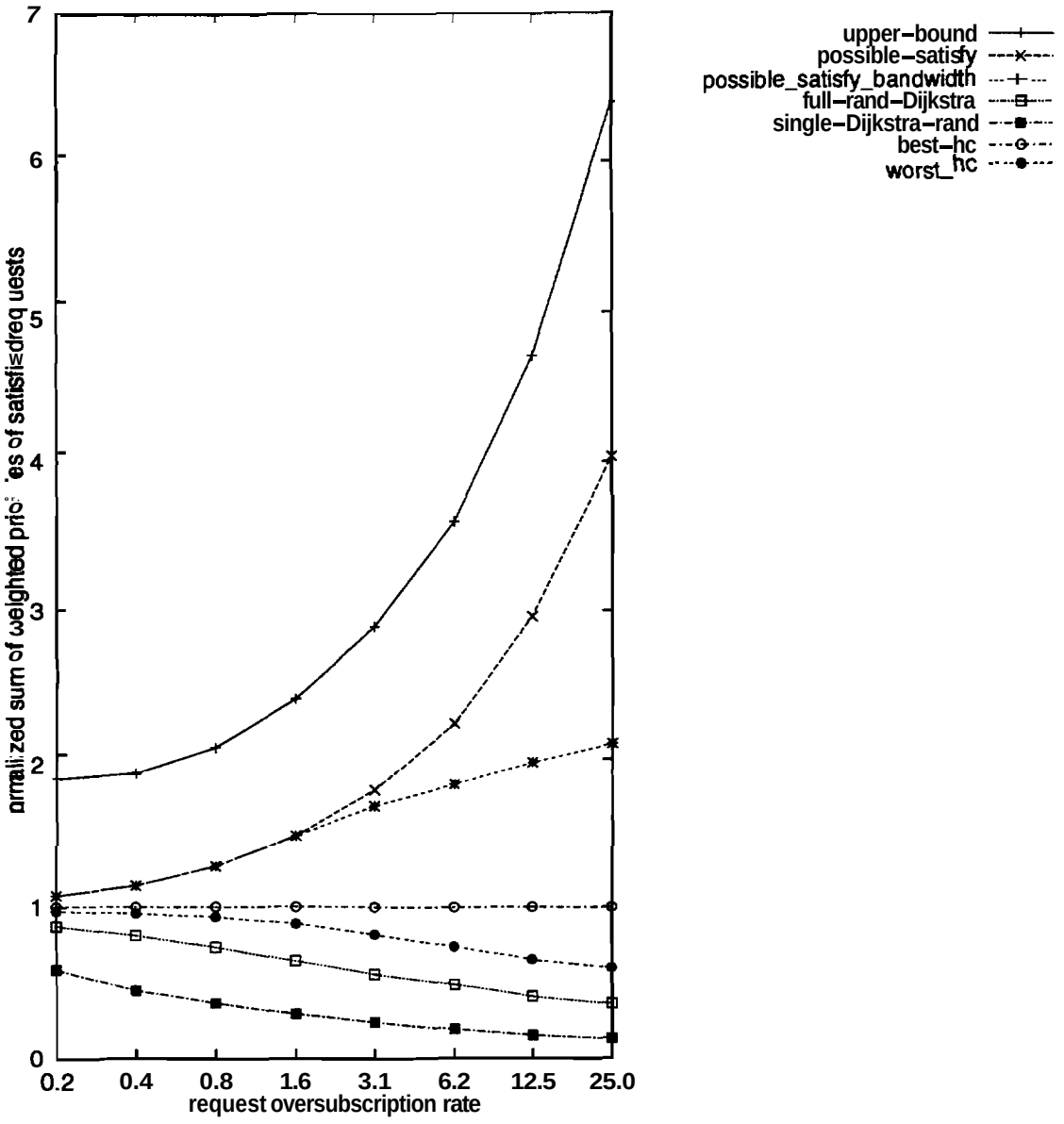

Figure 8.6: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an ω value of 2.

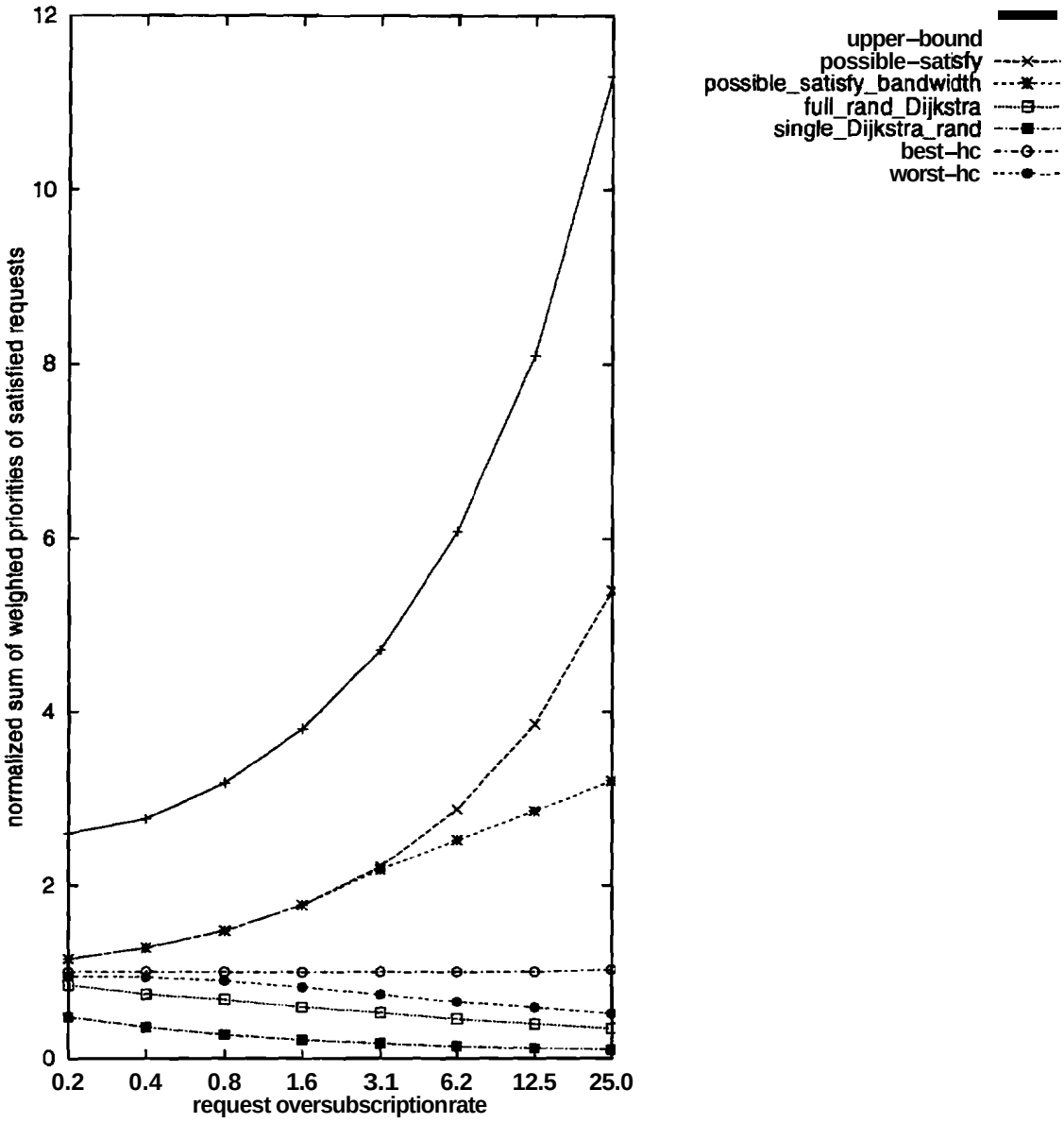

Figure 8.7: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an ω value of 2.

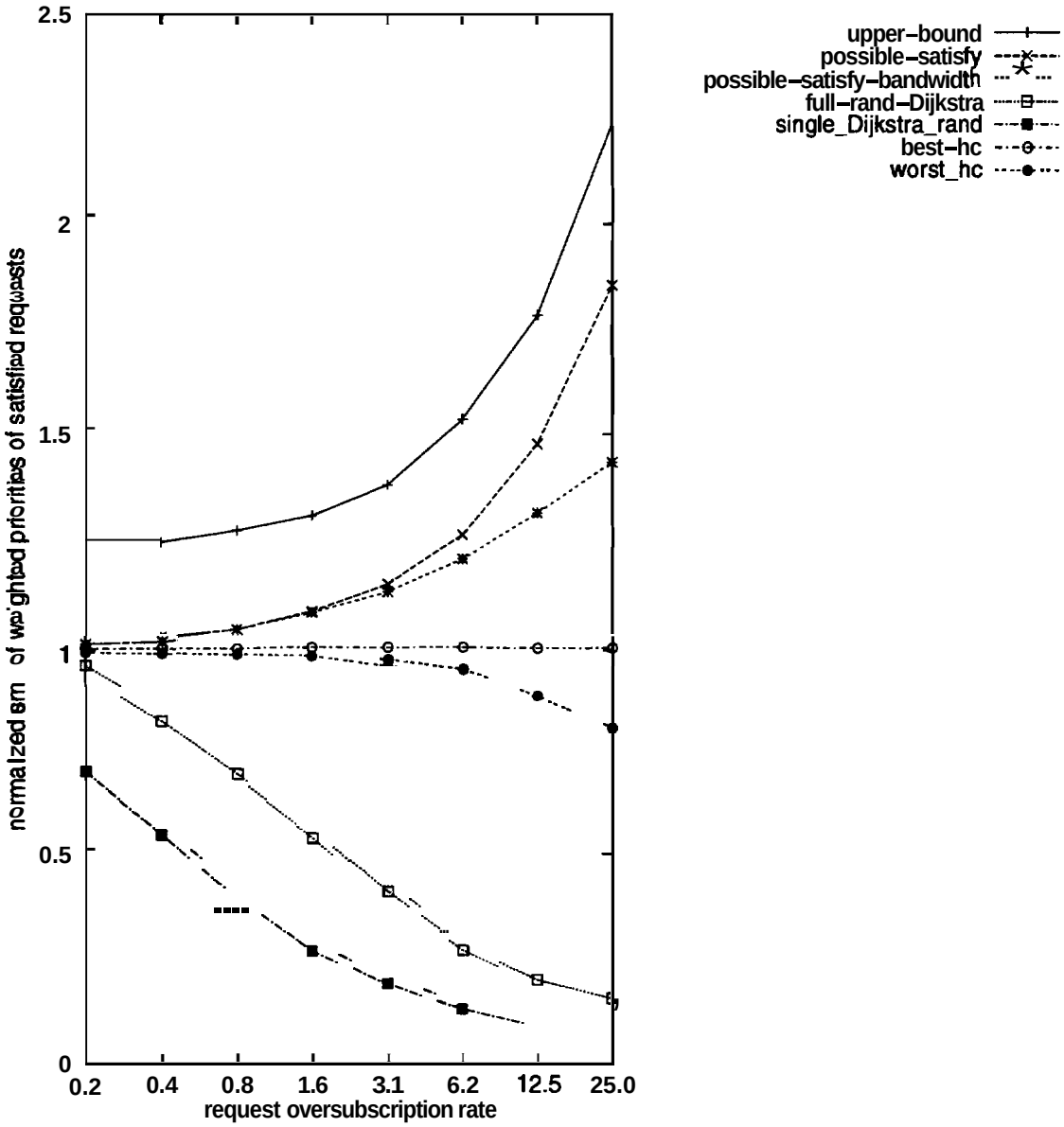

Figure 8.8: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full_one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an *w* value of 4.

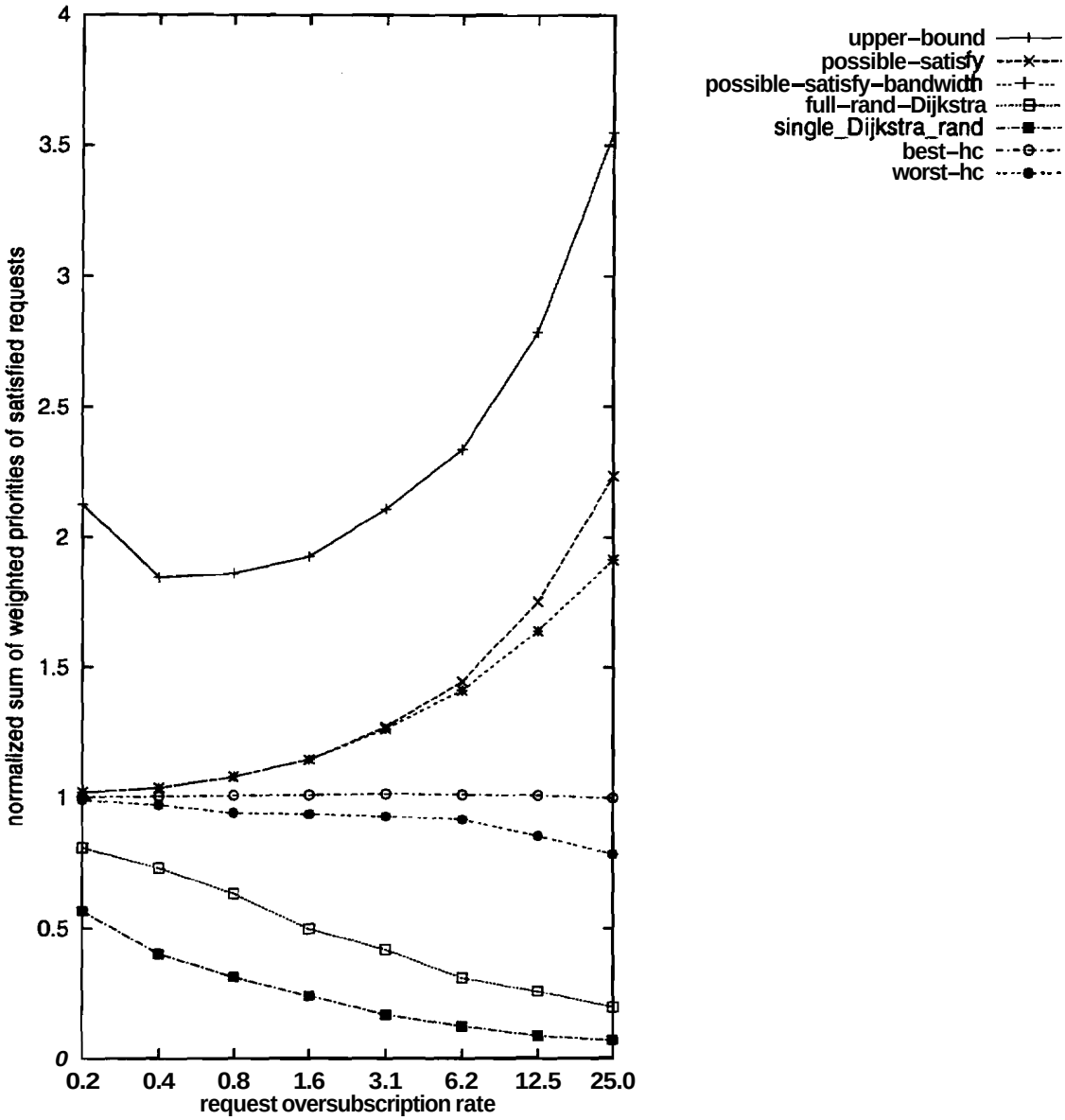

Figure 8.9: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an ω value of 4.

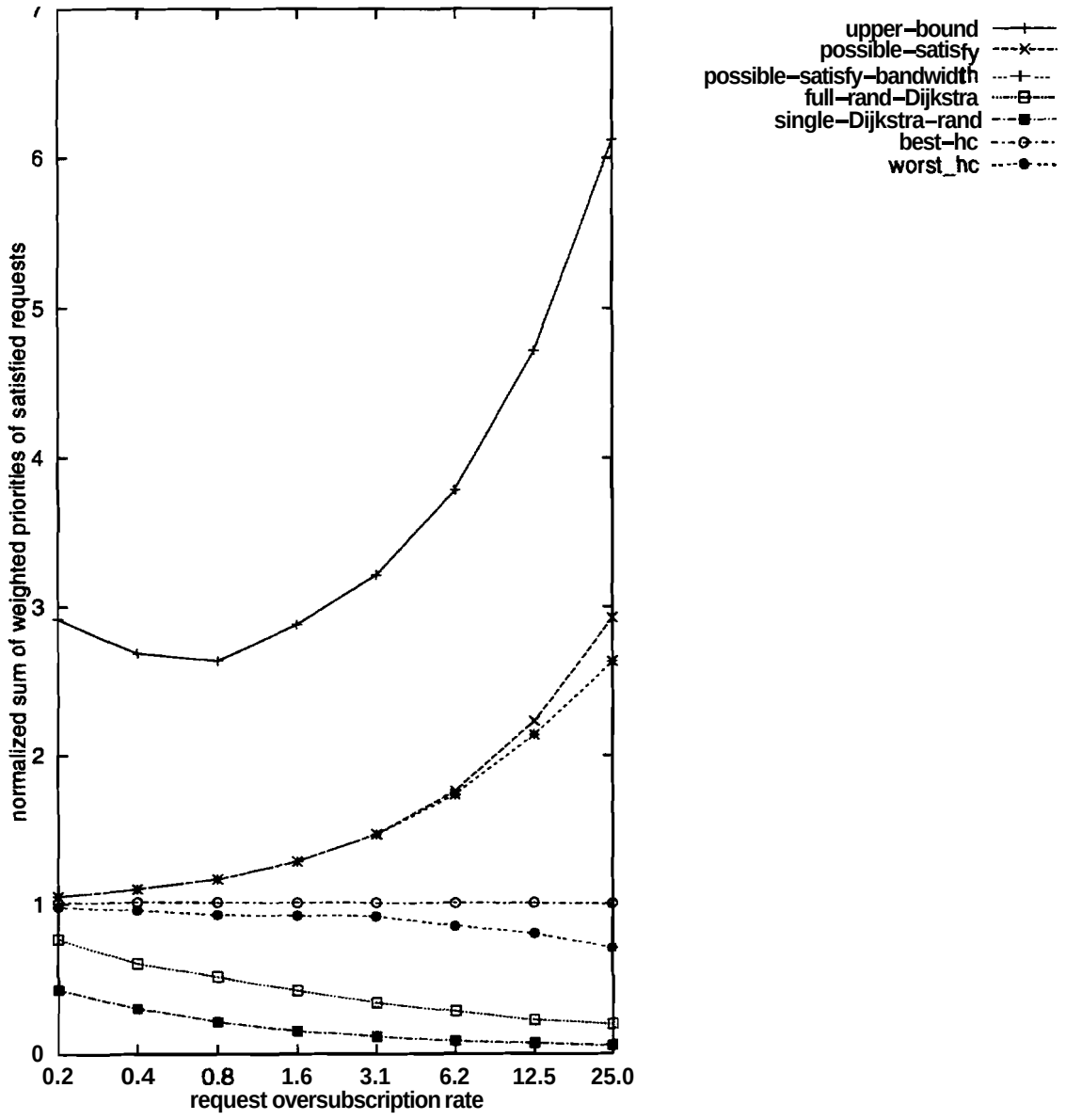

Figure 8.10: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an ω value of 4.

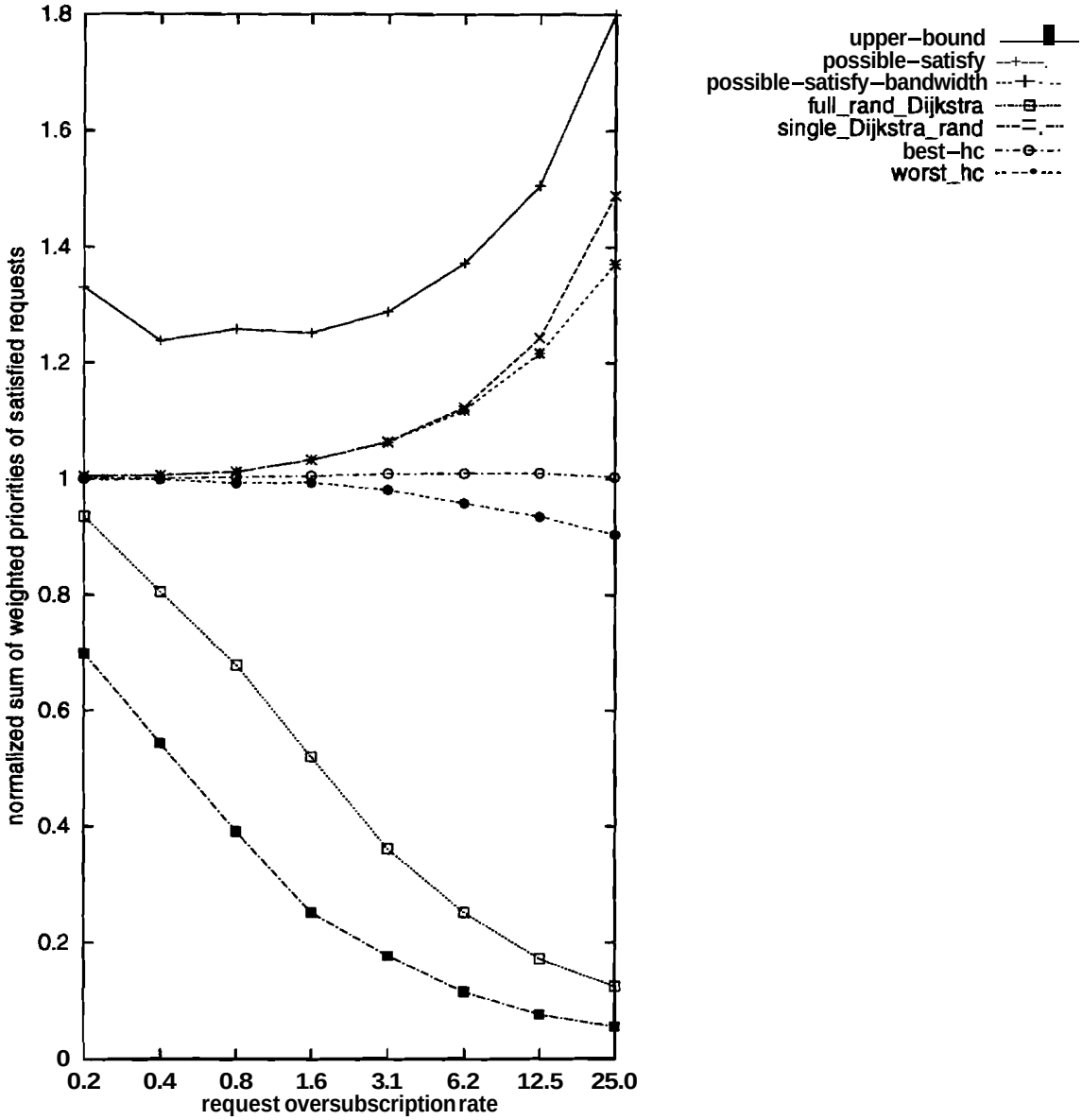

Figure 8.11: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an w value of 8.

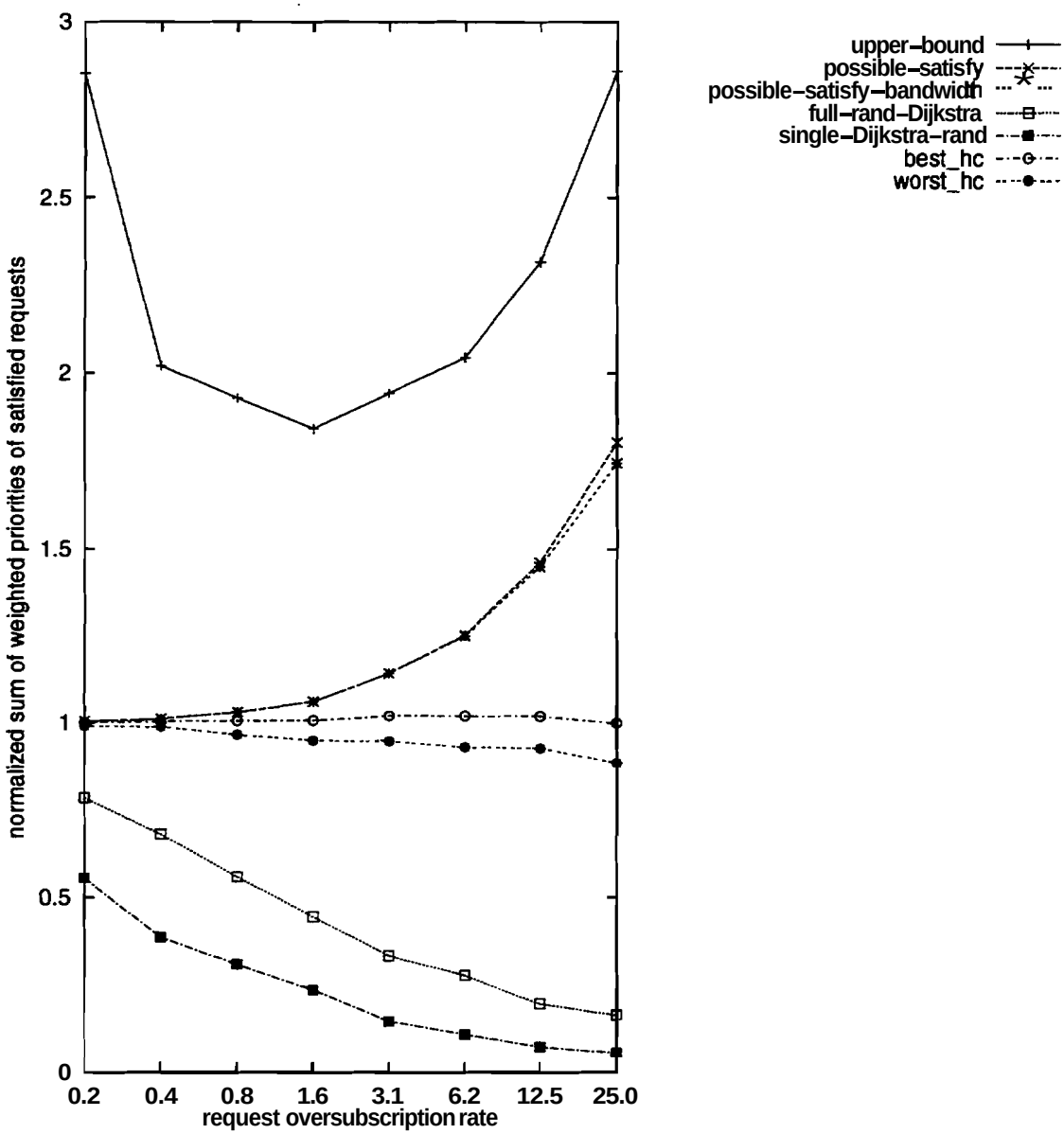

Figure 8.12: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one_C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an w value of 8.

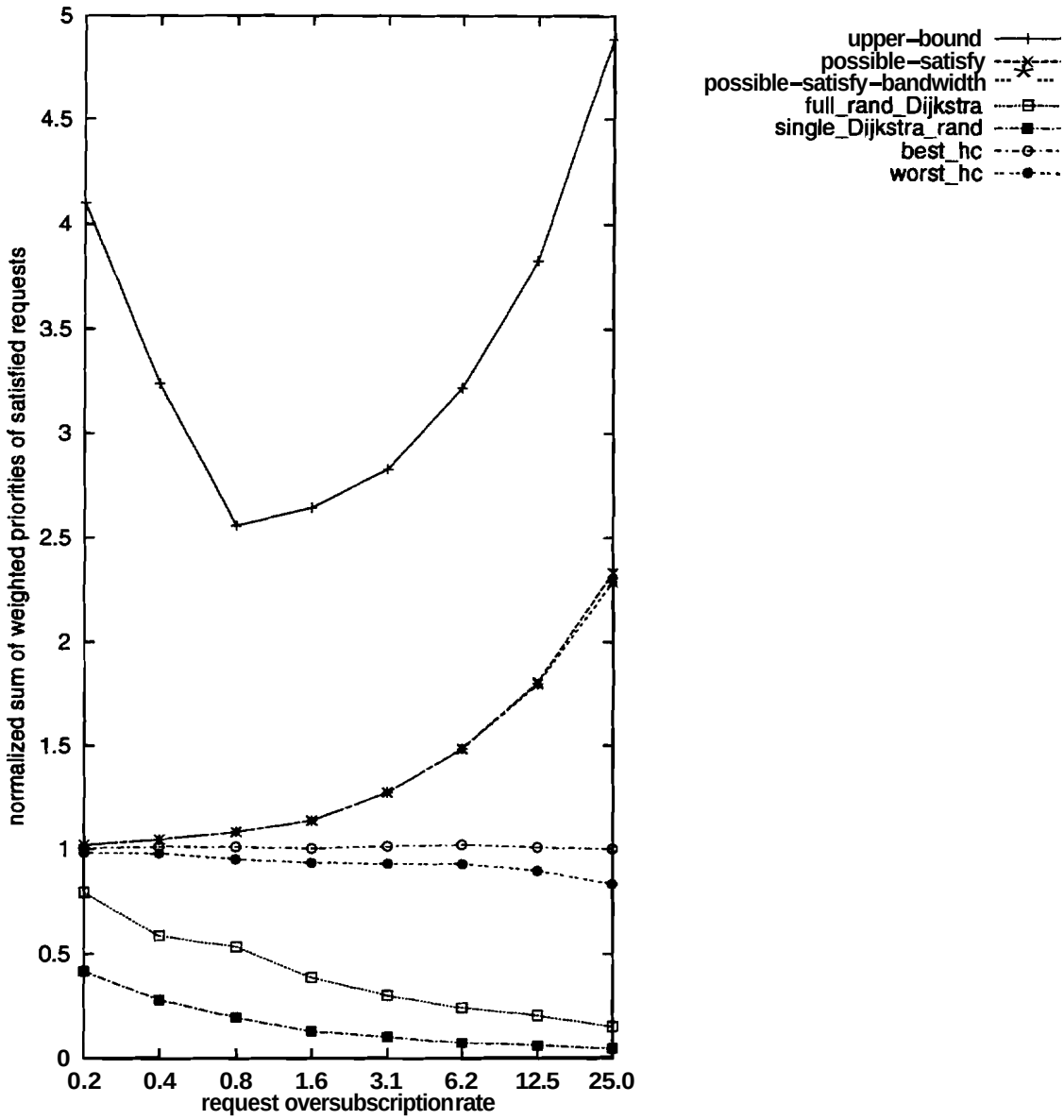

Figure 8.13: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an w value of 8.

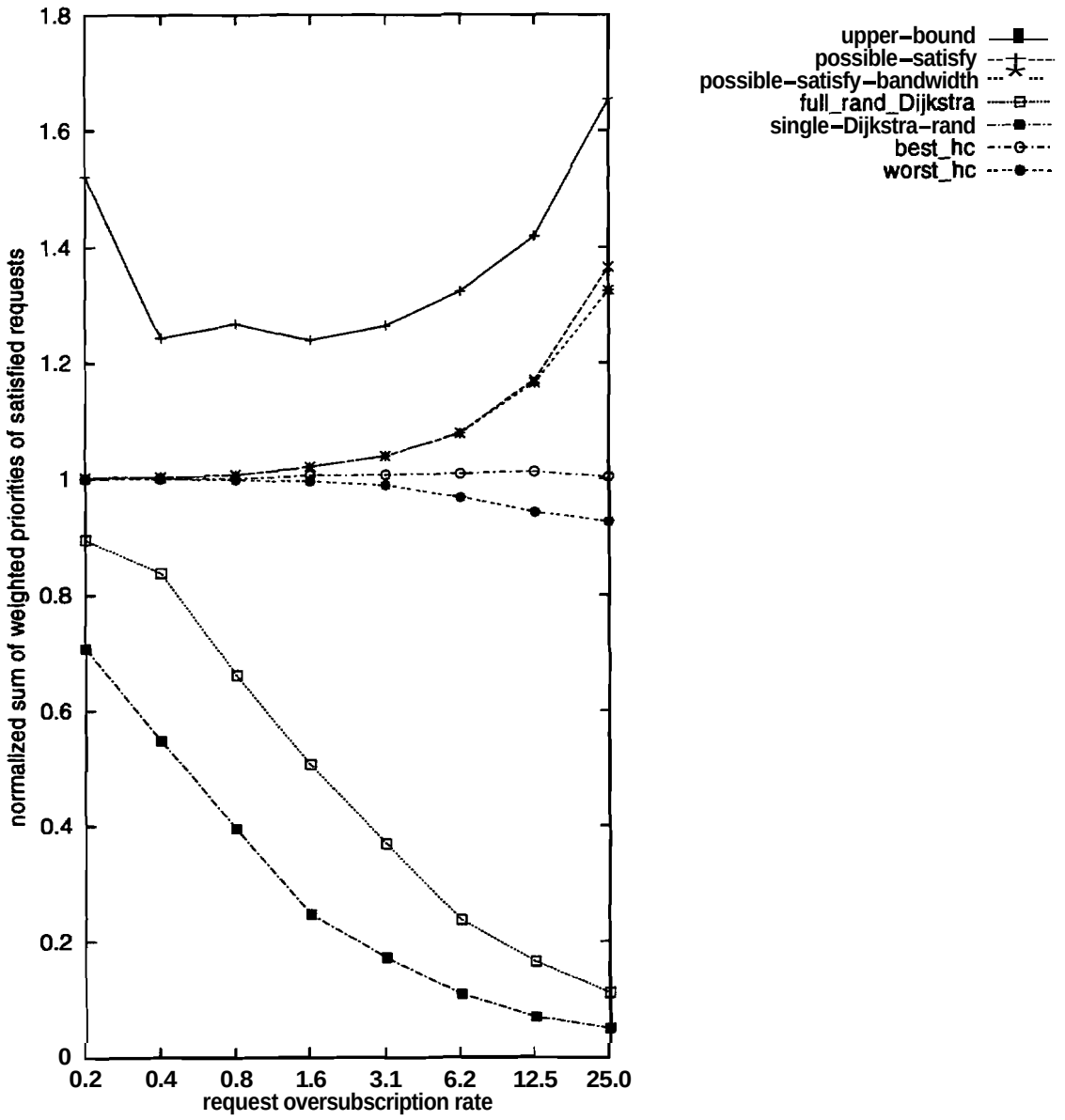

Figure 8.14: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an w value of 16.

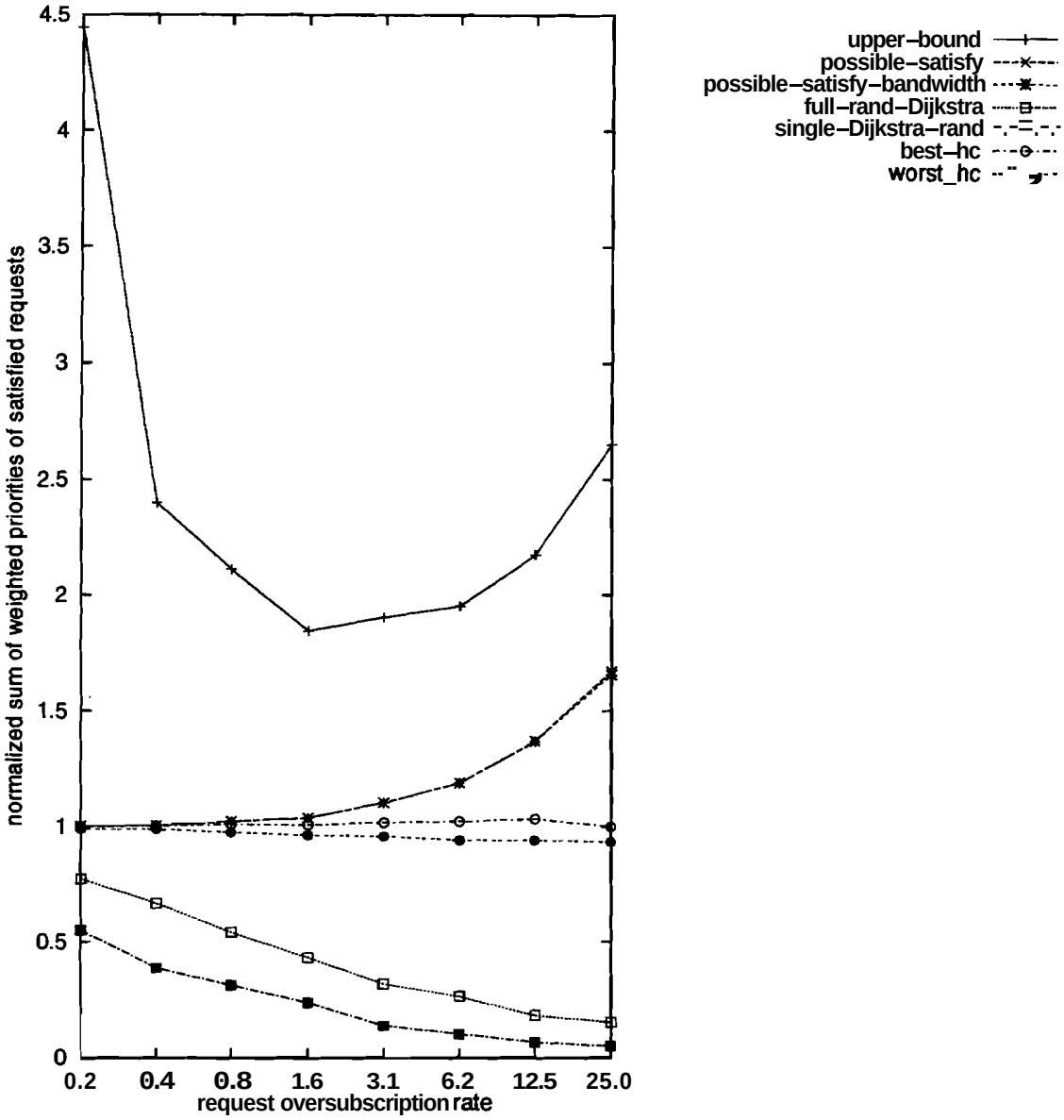

Figure 8.15: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an w value of 16.

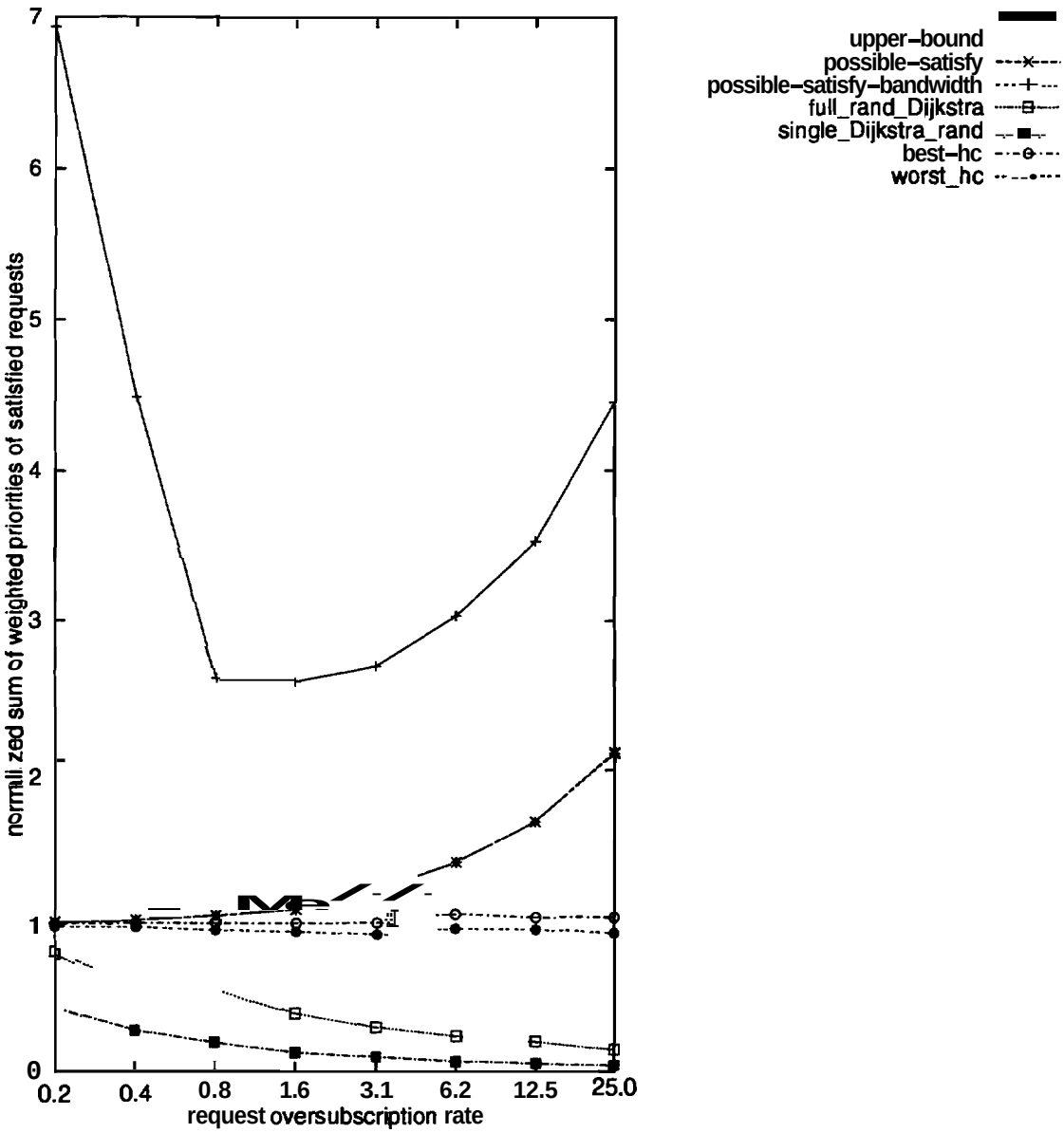

Figure 8.16: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full—one—C4. Shown are the upper bounds, lower bounds, and the performance of the best and worst heuristic for each oversubscription rate. The data sets had an average link traversal count of **3.5** and an *w* value of 16.

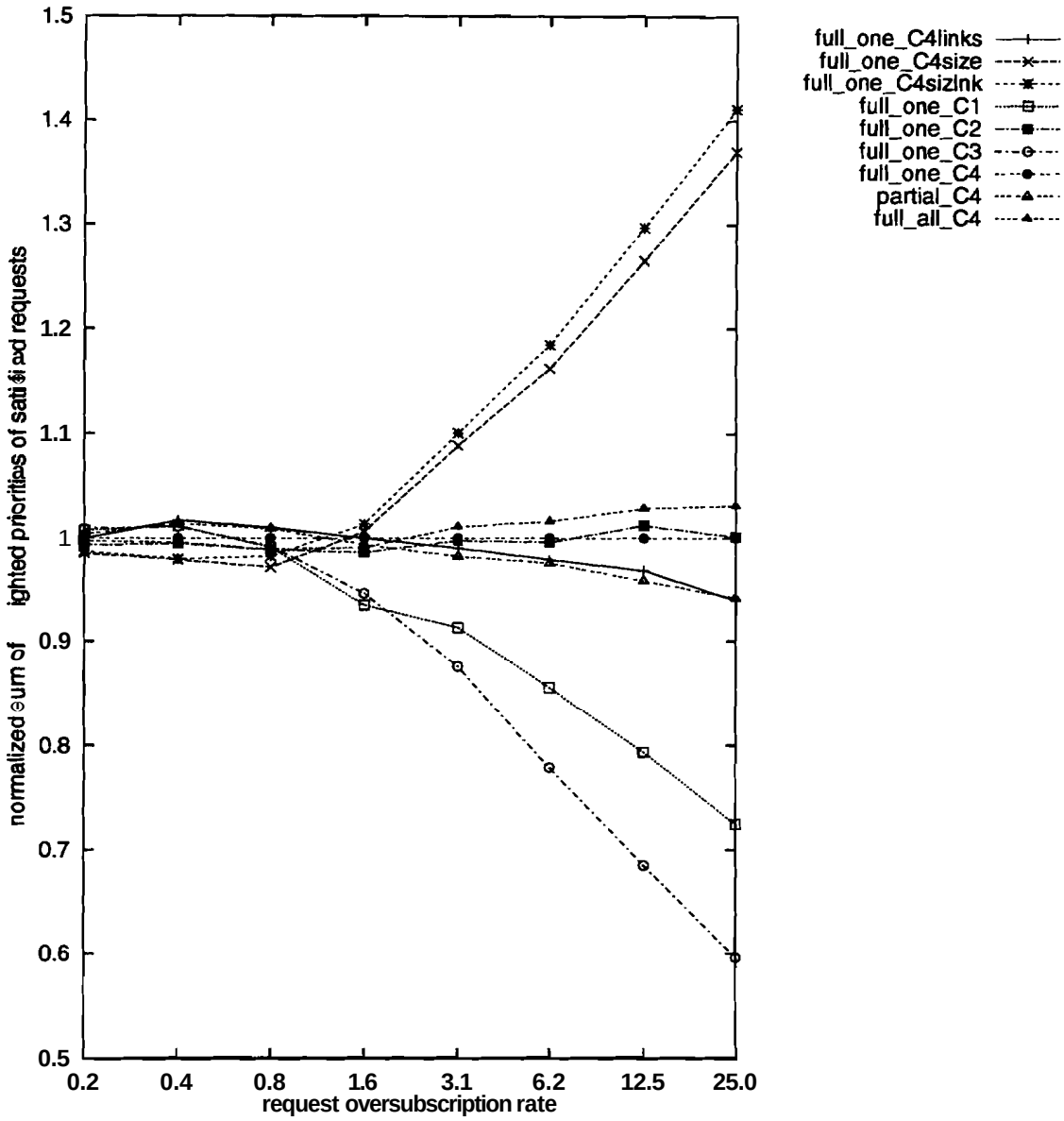

Figure 8.17: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an ω value of 1.

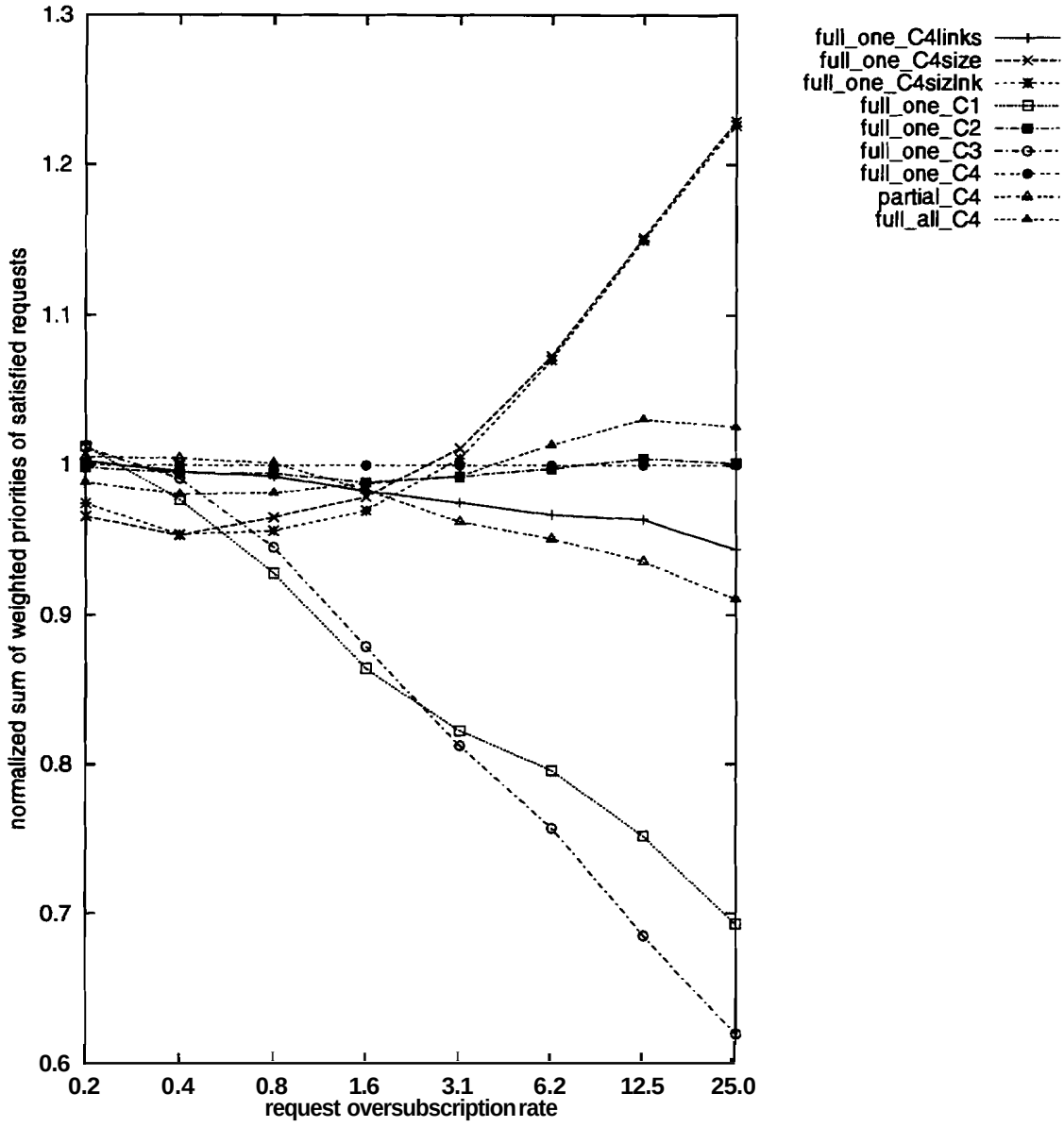

Figure 8.18: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an w value of 1.

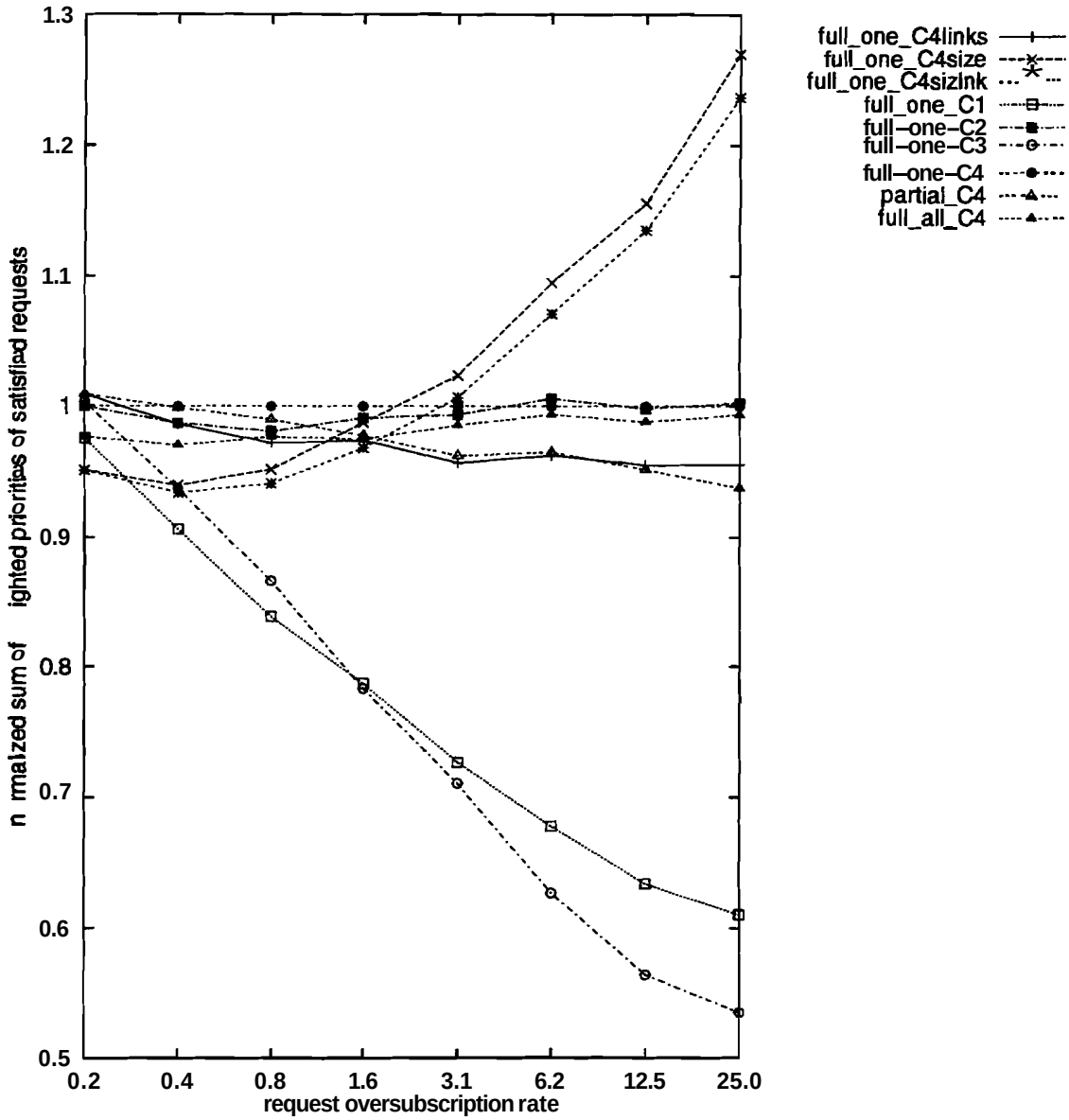

Figure 8.19: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an w value of 1.

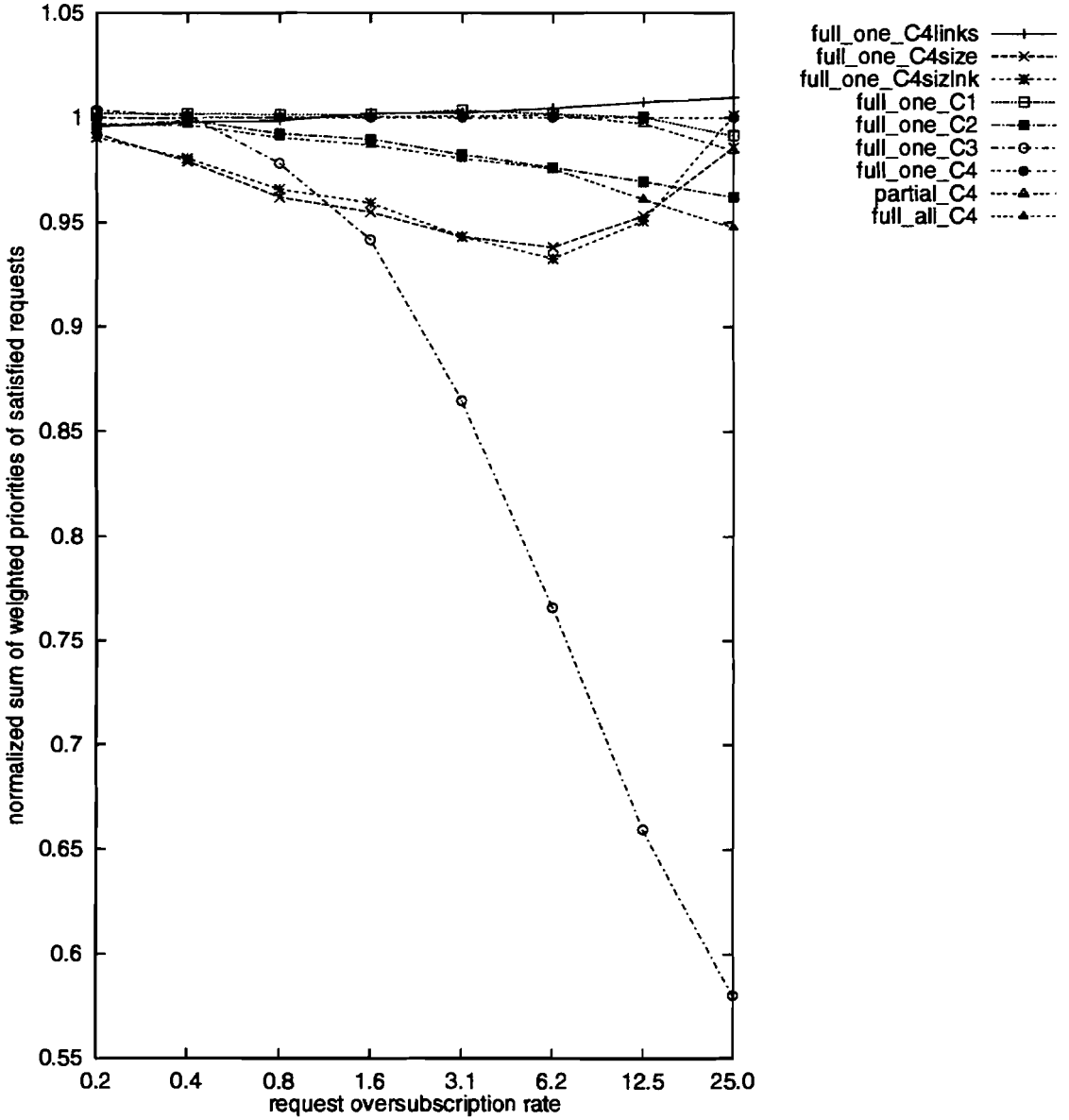

Figure 8.20: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an *w* value of 2.

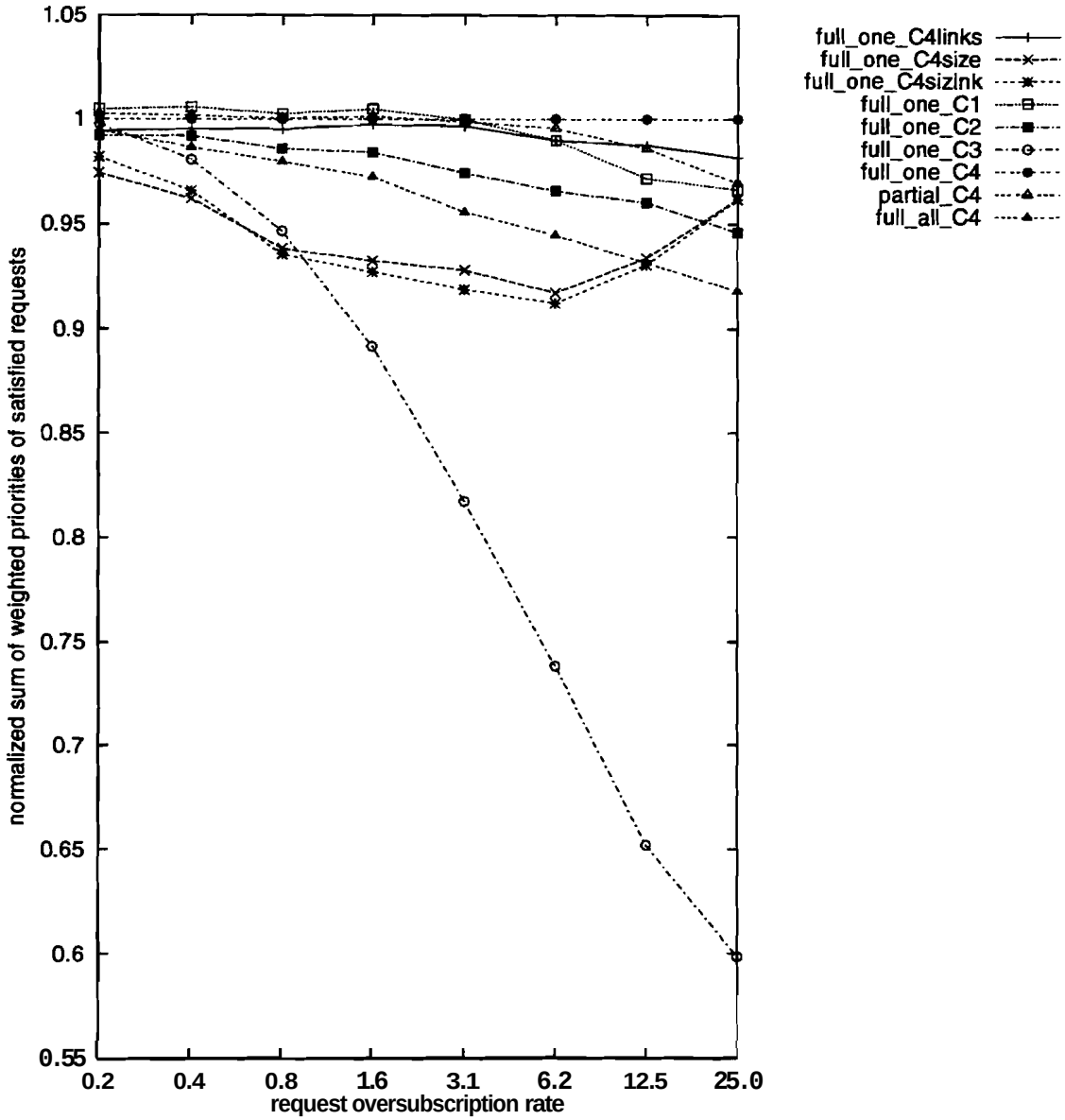

Figure 8.21: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an *w* value of 2.

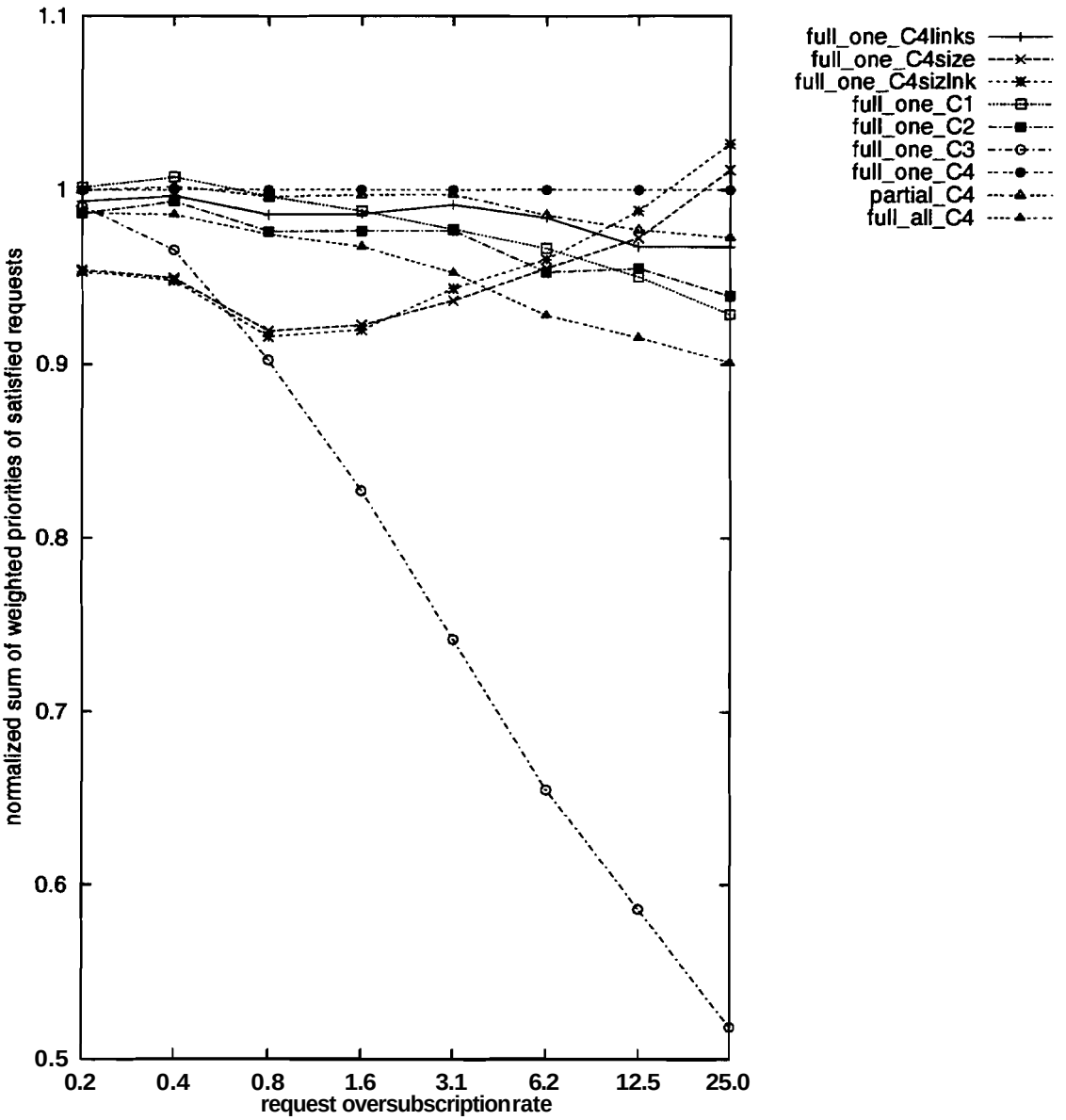

Figure 8.22: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an w value of 2.

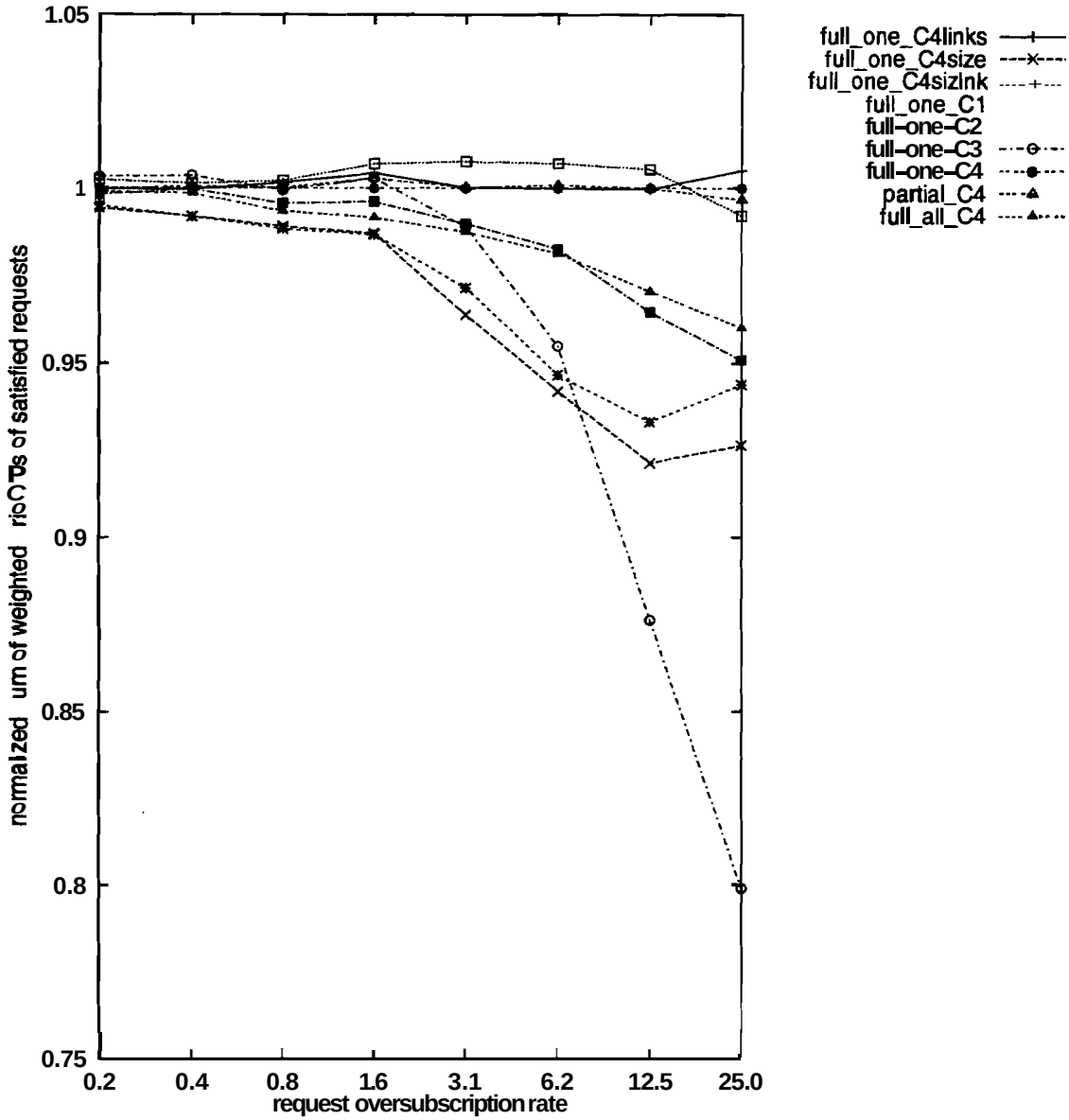

Figure **8.23:** Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an ω value of 4.

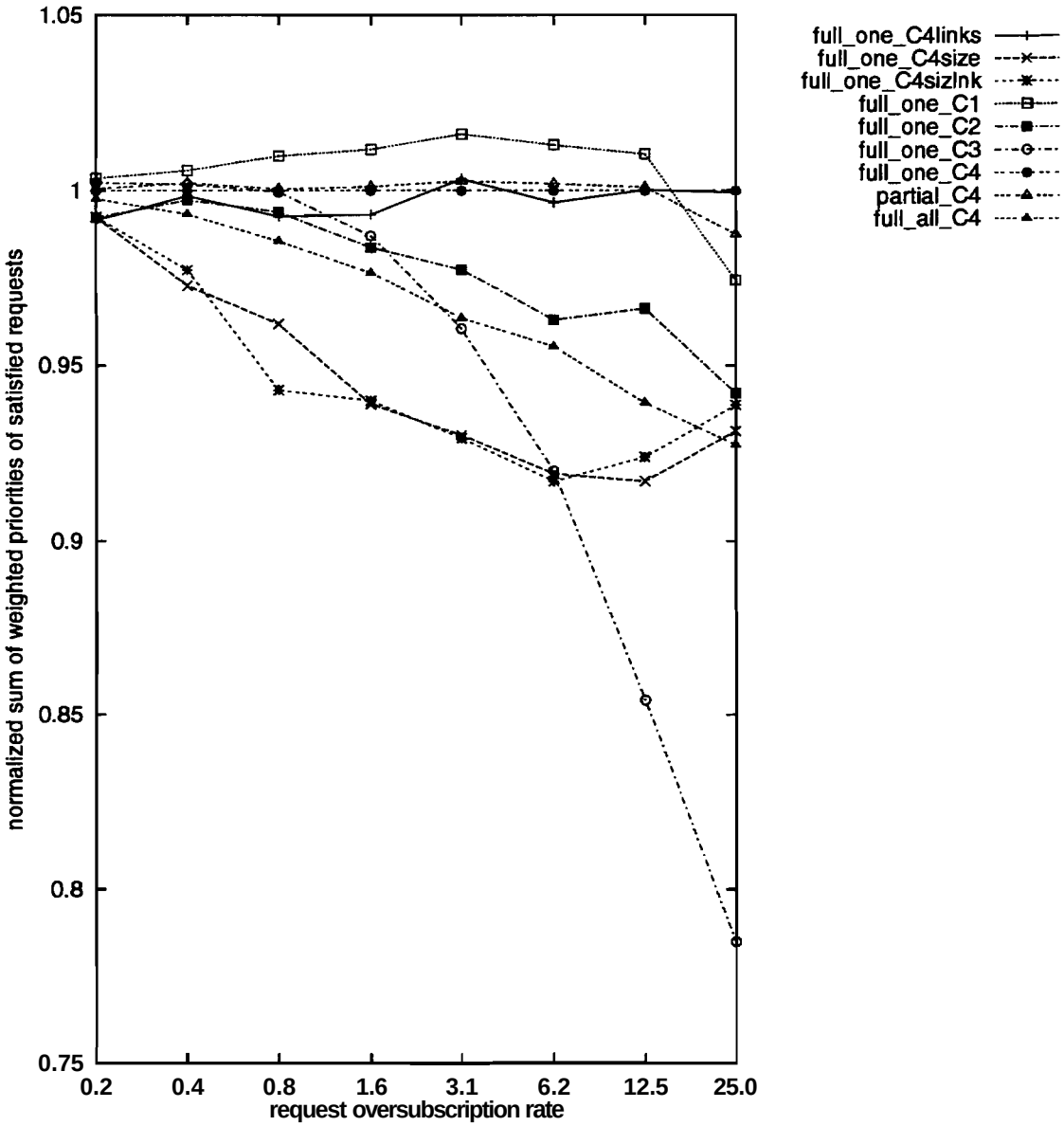

Figure 8.24: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an *w* value of 4.

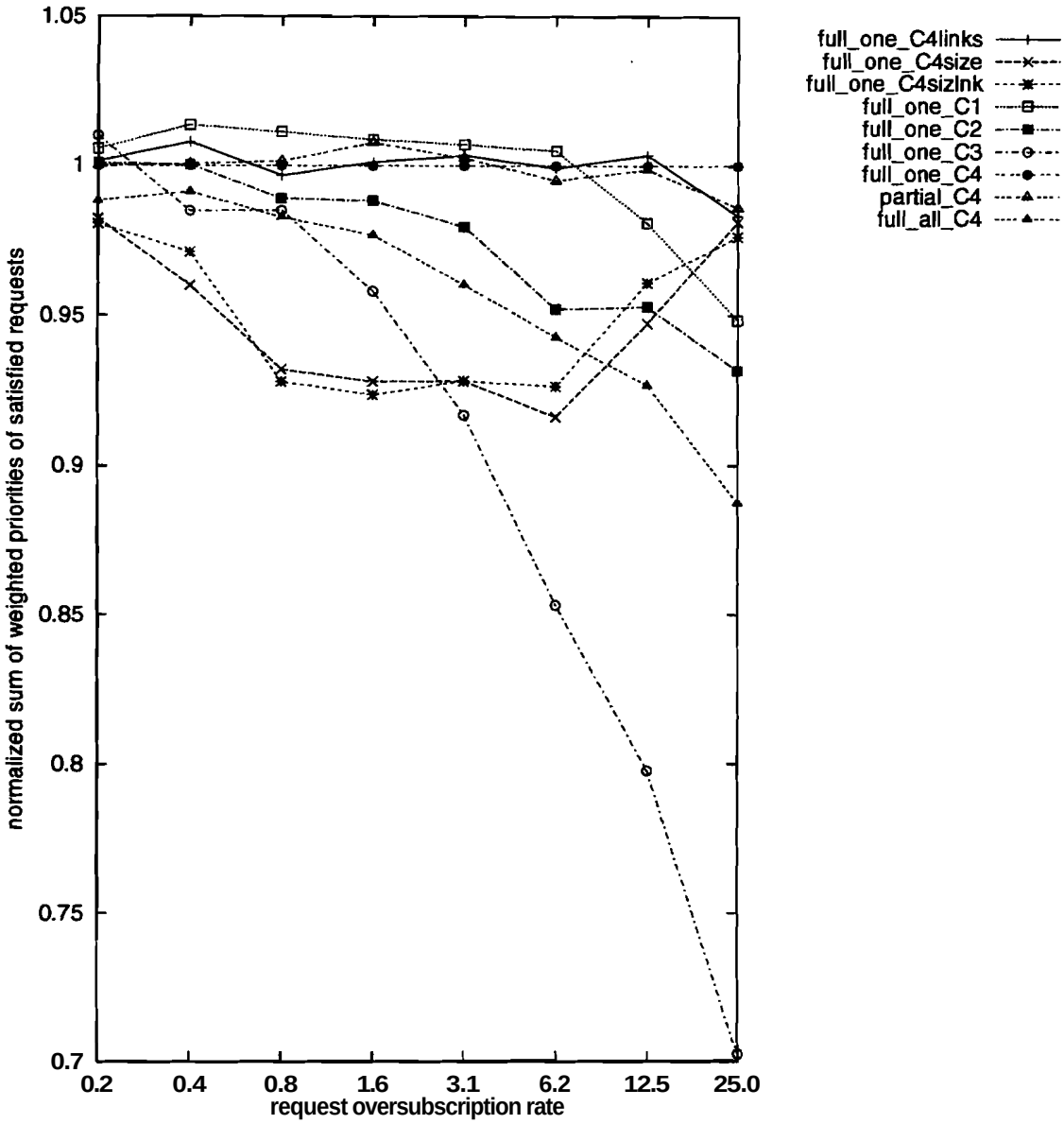

Figure 8.25: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an *w* value of 4.

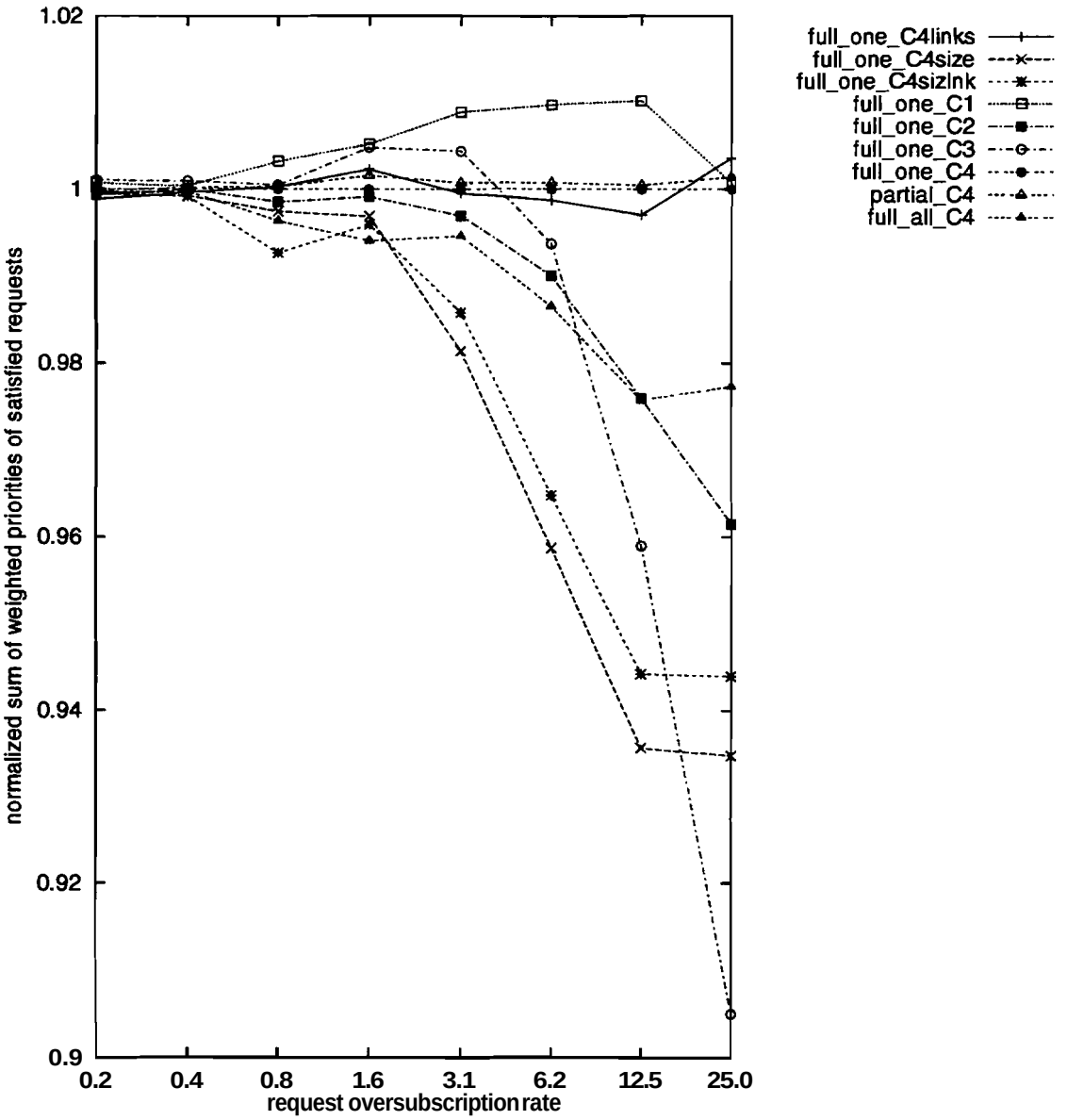

Figure 8.26: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an *w* value of 8.

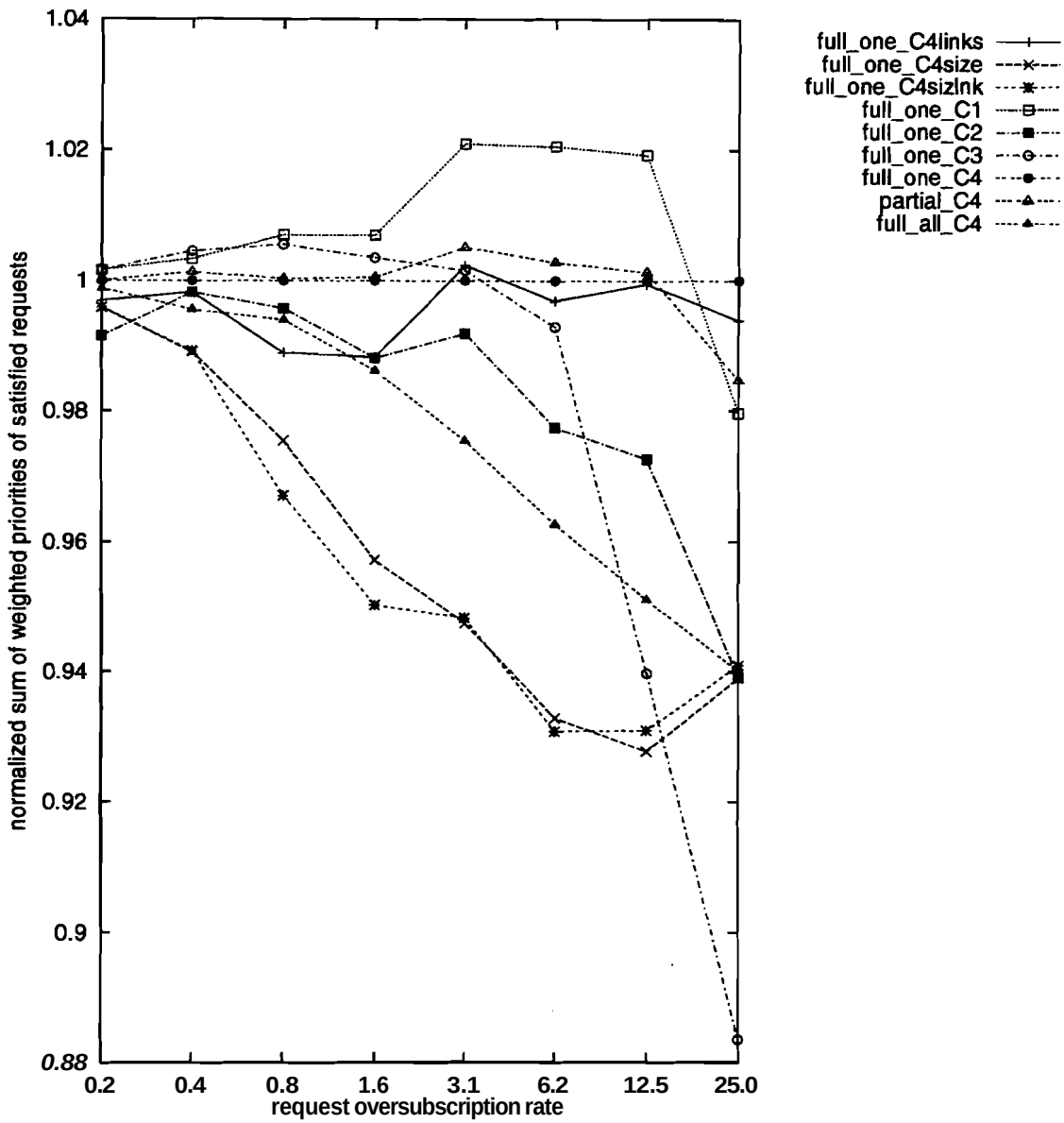

Figure 8.27: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an w value of 8.

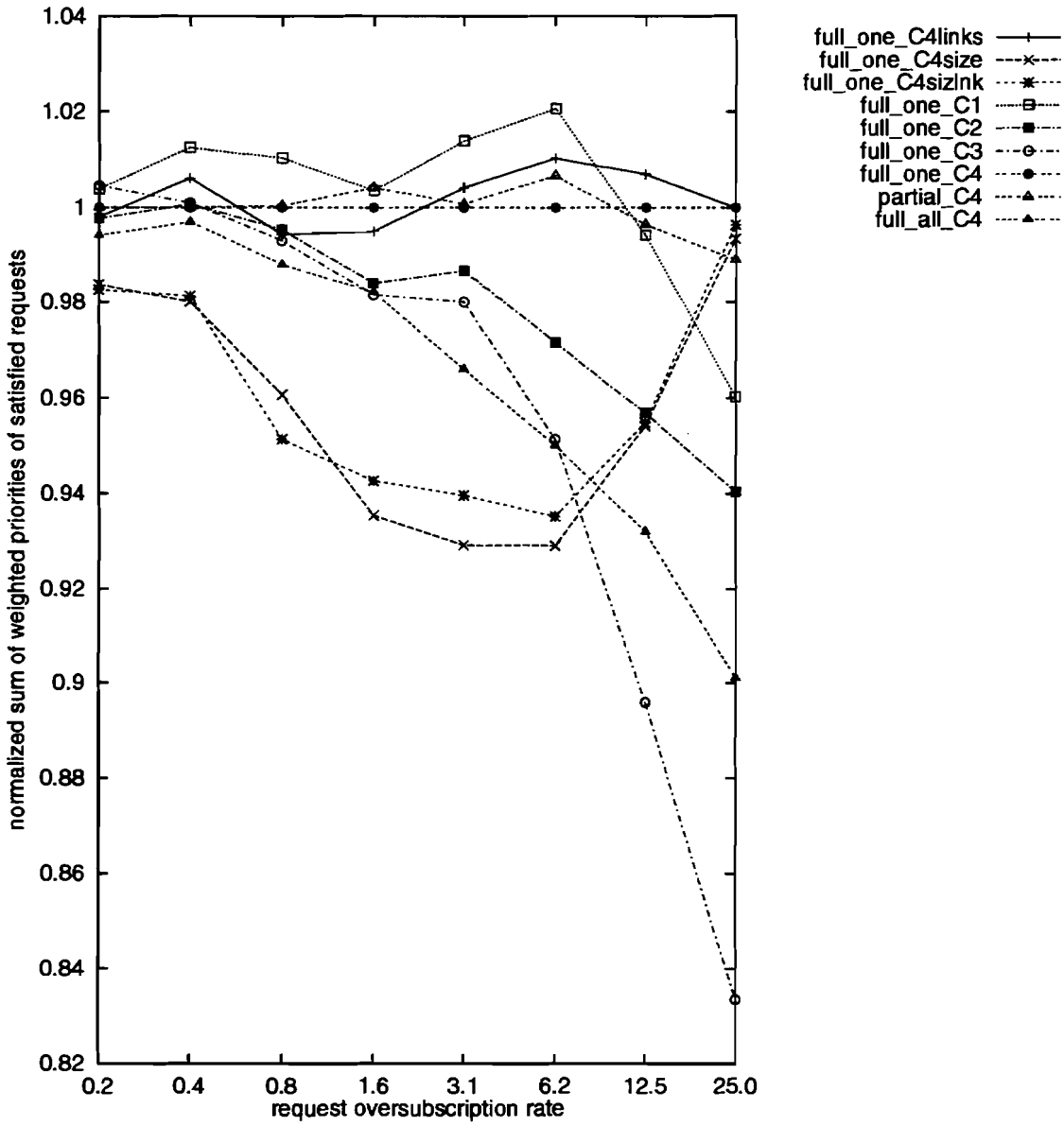

Figure 8.28: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an ω value of 8.

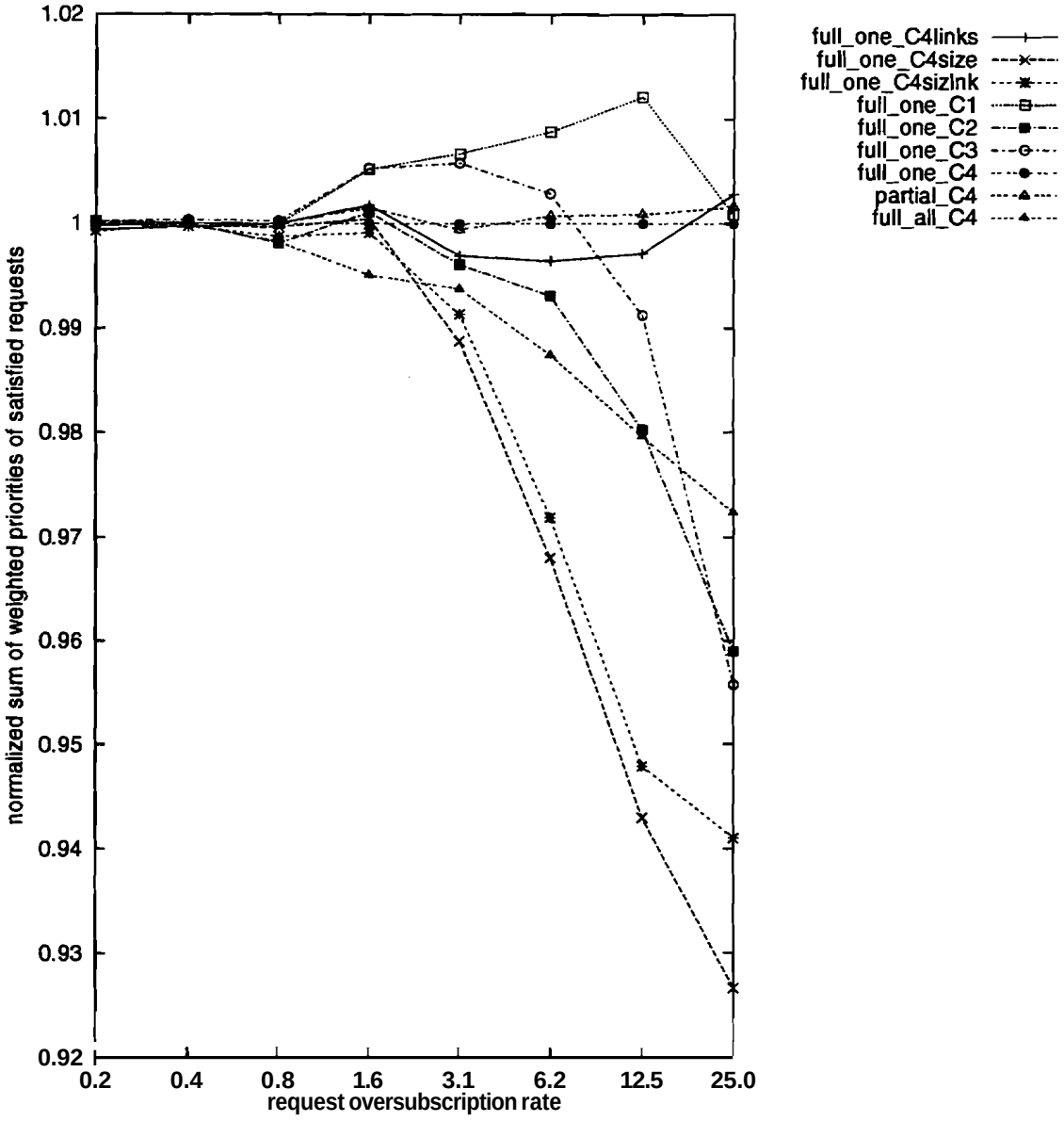

Figure 8.29: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 1.5 and an ω value of 16.

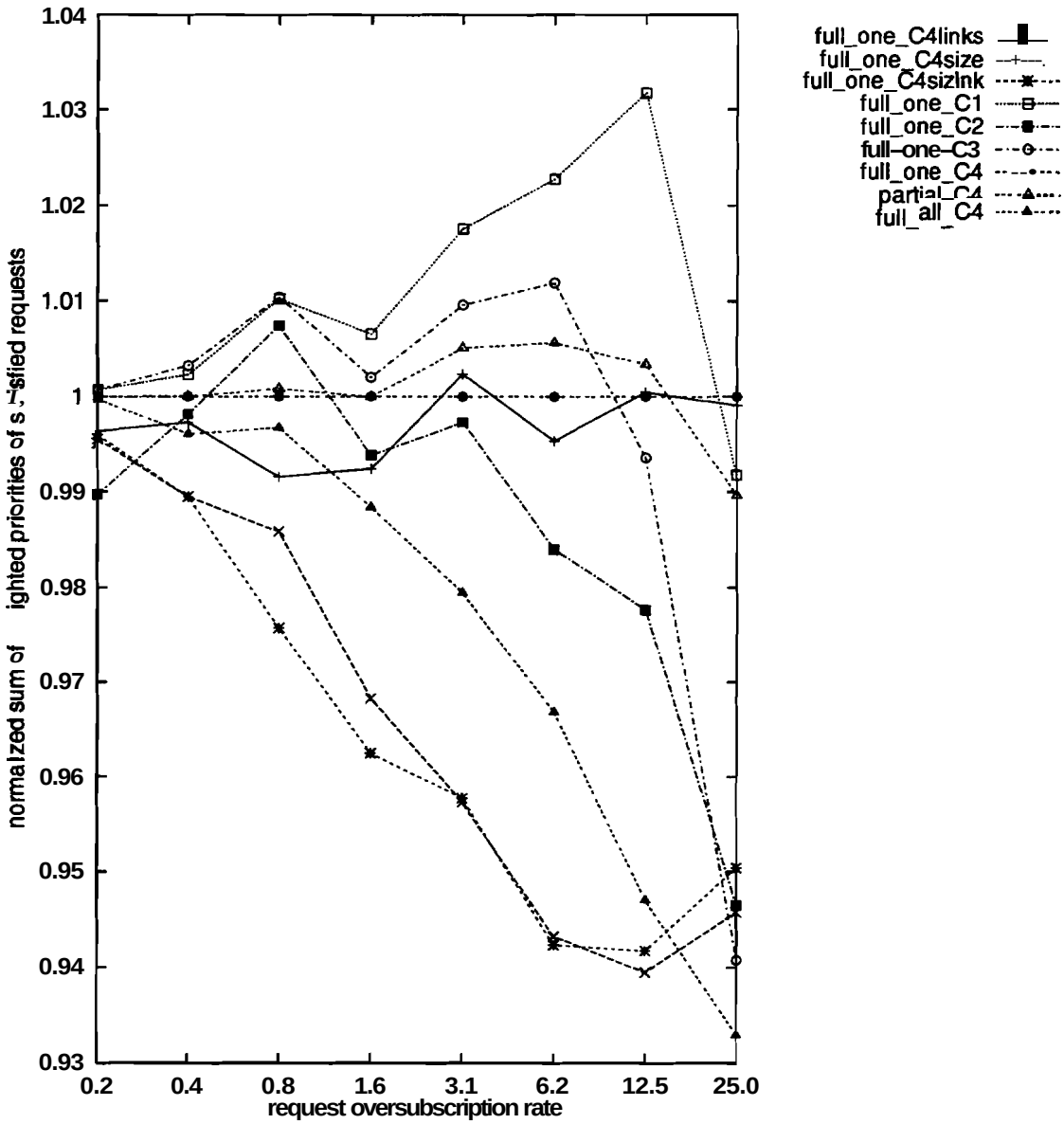

Figure 8.30: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full–one–C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 2.5 and an *w* value of 16.

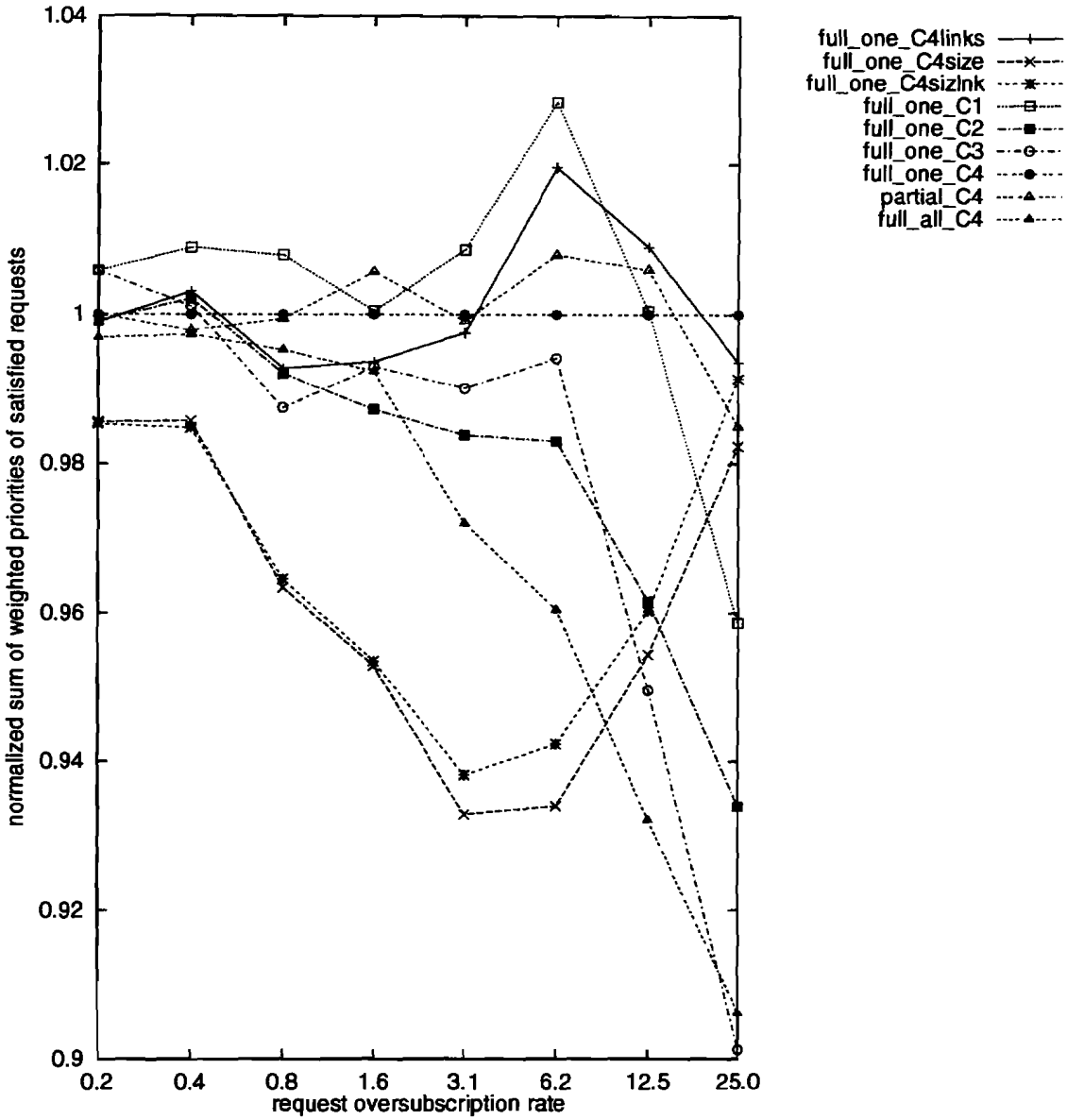

Figure 8.31: Weighted sum of satisfied requests' priorities normalized at each oversubscription rate to the performance of full−one−C4. Shown are all heuristics for each oversubscription rate. The data sets had an average link traversal count of 3.5 and an *W* value of 16.

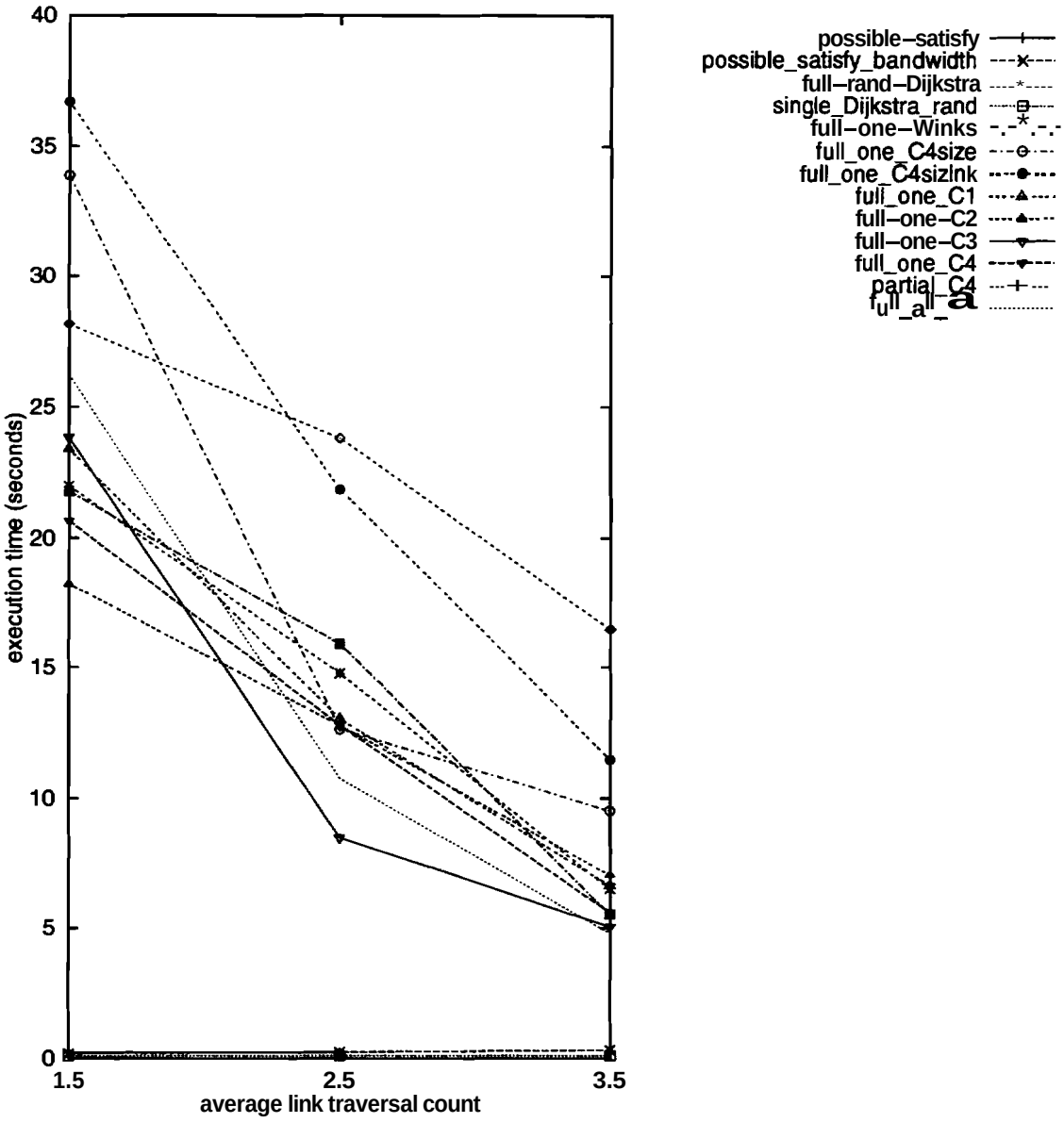

Figure 8.32: Average execution times of the heuristics and bounds for a data set with an oversubscription rate of 3.1 and $\omega = 4$. Times are in seconds on a four-processor 200 MHz Pentium Pro with 256 MB RAM.

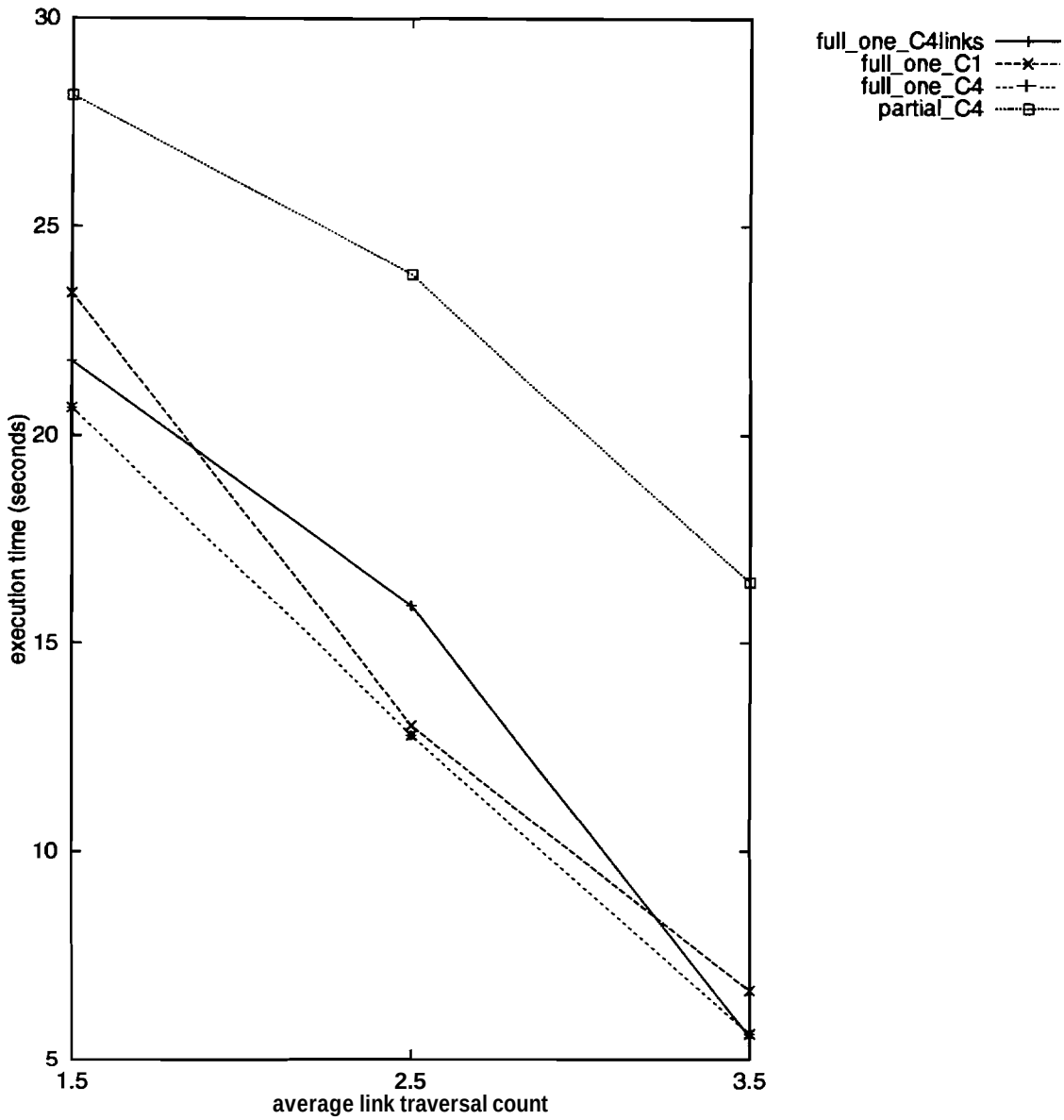

Figure 8.33: Average execution times of the four best-performing heuristics for a data set with an oversubscription rate of 3.1 and $w = 4$. Times are in seconds on a four-processor 200 MHz Pentium Pro with 256 MB RAM.

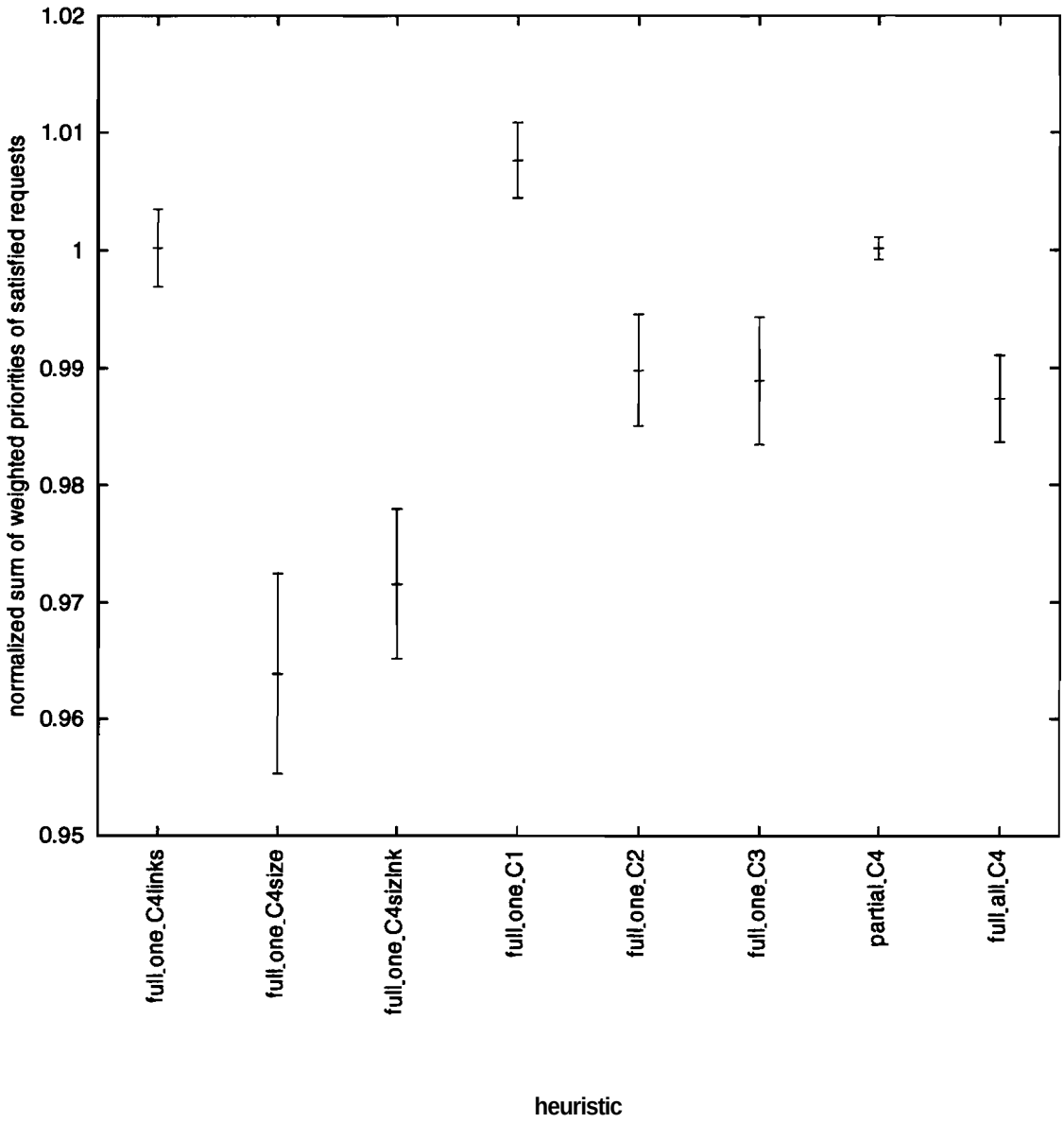

Figure 8.34: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4. Shown are 95% confidence intervals for all heuristics (except full−one−C4) in a data set with an average link traversal count of 1.5 and an w value of 4.

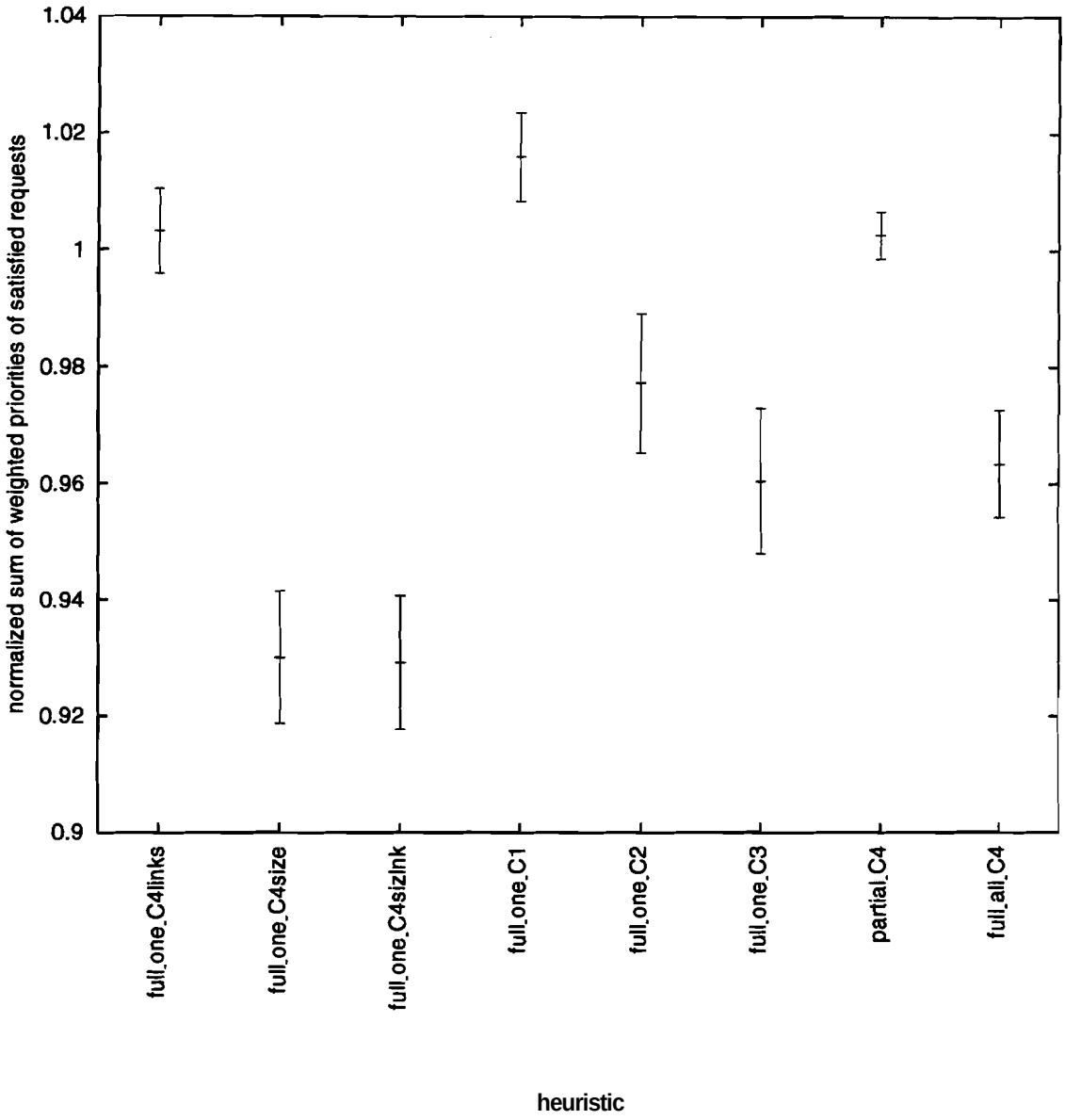

Figure 8.35: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4. Shown are 95% confidence intervals for all heuristics (except full−one−C4) in a data set with an average link traversal count of 2.5 and an *w* value of 4.

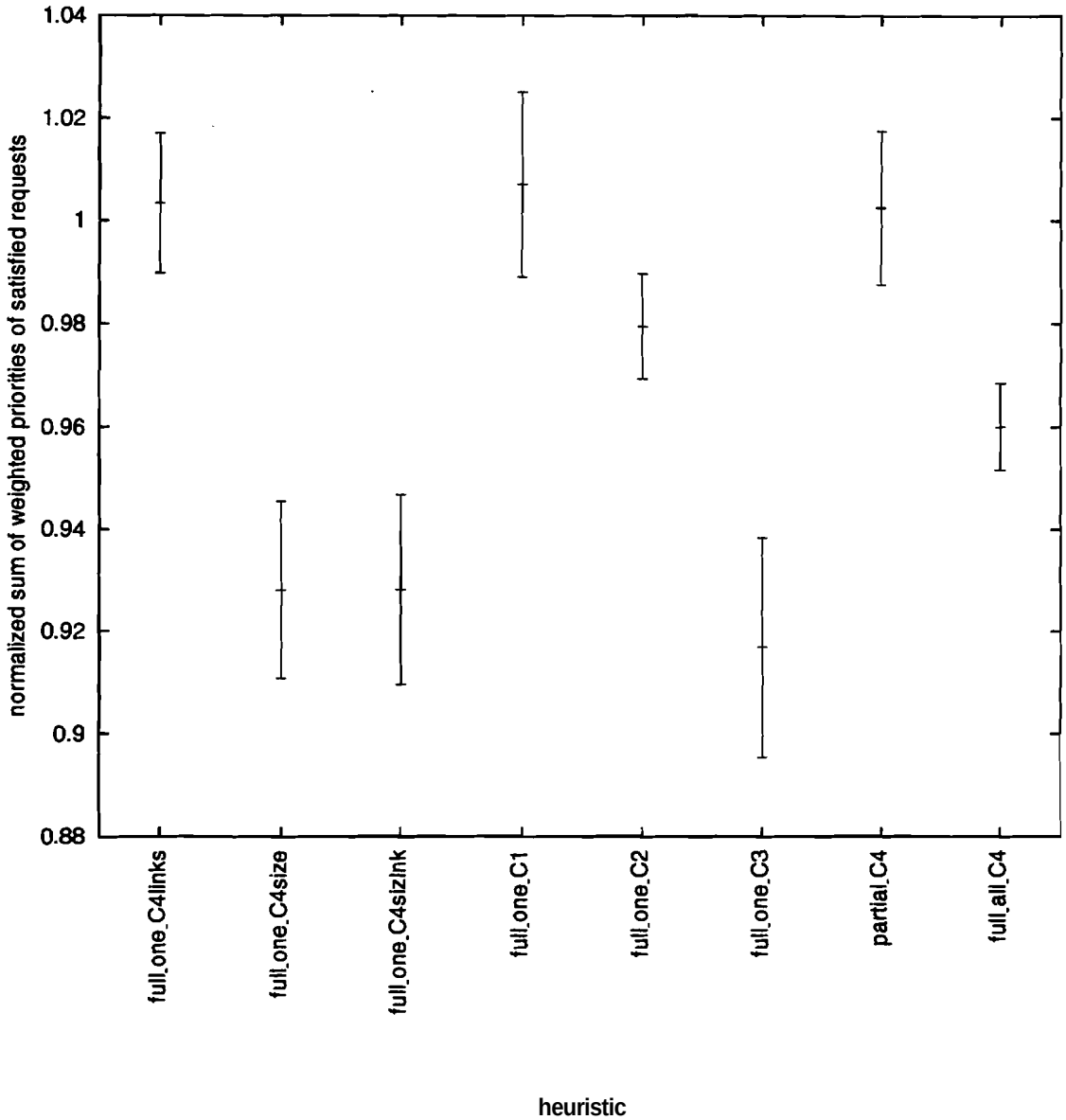

Figure 8.36: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4. Shown are 95% confidence intervals for all heuristics (except full−one−C4) in a data set with an average link traversal count of 3.5 and an w value of 4.

Table 8.4
Number requests satisfied at each priority level by full_one_C4 with an average link traversal count of 2.5 and an oversubscription rate of 1.6. The "level by level" column shows the effect of allocating resources for all priority class α requests before all priority class β requests where $\alpha > \beta$.

priority level	number requests satisfied					
	ω					level by level
	1	2	4	8	16	
5	4.2	8.3	8.4	8.4	8.4	8.2
4	9.8	16.0	16.1	16.1	16.1	16.0
3	16.4	23.4	23.4	23.1	23.5	22.8
2	28.8	33.8	31.2	27.0	32.5	32.5
1	57.6	50.5	44.8	43.1	43.0	50.1
0	118.8	81.6	82.0	85.9	84.9	78.6

Table 8.5
Number requests satisfied at each priority level by full—one—C4 with an average link traversal count of 2.5 and an oversubscription rate of 6.2. The "level by level" column shows the effect of allocating resources for all priority class α requests before all priority class β requests where $\alpha > \beta$.

priority level	number requests satisfied					
	ω					level by level
	1	2	4	8	16	
5	9.7	30.4	31.3	31.2	31.2	30.4
4	19.3	44.4	45.3	45.2	45.0	43.6
3	34.1	53.4	51.3	51.2	50.8	52.0
2	58.4	63.0	54.8	48.1	41.4	60.6
1	117.4	94.2	74.4	70.2	69.1	87.3
0	241.5	134.9	126.3	133.5	139.3	122.7

9. Data Items With Multiple Versions

9.1 Approach

In this section, a variable time, variable accuracy algorithm will be presented to deal with data items with a higher quality and lower quality version, as mentioned in Section 3. The higher quality data item is assumed for simplicity to be twice the size of the lower quality data item. The higher quality data item, however, has four times as much "worth" to the end user as the lower quality data item. This worth was chosen to indicate that the system should be penalized for selecting the lower quality data item over the higher one. The lower quality data item thus has half of the worth per byte of the higher quality data item.

The approach used to incorporate these lower quality data item versions into the developed heuristics was to create an iterative algorithm that attempts to create a new schedule S_h with each iteration that has a smaller effect $E[S_h]$. In the first iteration, only the higher quality versions of the data items are considered satisfiable by the value $Sat[i, \mathrm{k}]$ (where $0 \leq \mathrm{i} < 2\rho$ and $0 < \mathrm{k} \leq Nrq[i]$). That is, $Sat[i, \mathrm{k}]$ (from the cost criteria of Section 5) can only be 1 if $0 \leq i < \mathrm{p}$. A heuristic is then used with Dijkstra[7]'s algorithm to create a complete schedule of data transfers, which corresponds to the research described in Section 8.

After the first iteration schedule has been determined, the value of $Sat[j, \mathrm{k}]$ (where $0 \leq \mathrm{j} < \mathrm{p}$) for the second iteration is only allowed to be 1 if $Request[j, \mathrm{k}]$ was satisfied in the previous iteration. The value of $Sat[j + \mathrm{p}, \mathrm{k}]$ is then only allowed to be 1 if $Request[j, \mathrm{k}]$ was not satisfied in the previous iteration. A complete new schedule is created using a heuristic with Dijkstra[7]'s algorithm. That is, if during iteration one a requesting destination does not receive its higher quality requested data item, then in the second iteration, it will request the lower quality version of that data item instead. The schedule produced by the second iteration will then likely satisfy at least a few

lower quality data item requests (of higher priority) in place of higher quality data item requests (of lower priority). The higher quality data item requests that are not satisfied in the second iteration then request their respective lower quality versions for the third iteration. This iterative process can be repeated as many times as allotted execution time permits, and can stop at any time after the first iteration and output the best schedule that it has generated thus far. (This assumes that the best schedule is kept separately after each iteration and that the last iteration performance may not result in the best schedule.)

9.2 Evaluation of Simulations

The data sets used for these experiments were a subset of the data sets created for the simulation study of Section 8. Only the data sets with average link traversal counts of 2.5 were used. Five iterations of the variable accuracy algorithm were run. Results from those runs are shown in Figures 9.1 through 9.40. It should be noted that each graph is normalized to the performance of full−one−C4 at the end of its first iteration, which is the same as the performance of full−one−C4 in the study of Section 8. Figures 9.1 through 9.8 are included for comparison, but keep in mind that $w = 1$ is a degenerate case.

For less oversubscribed networks, the heuristics are almost all able to increase their own respective performance with additional iterations (for example, Figures 9.17, 9.18, 9.19, and 9.20). For more oversubscribed networks, this is not generally the case (for example, Figures 9.24, 9.32, and 9.40). All of the cost criteria used here except $C1$ consider more than one destination as part of the cost of sending a data item to its next machine. The implementation of the multiple versions approach works against this, particularly at higher oversubscription rates. This is because a data item that contributes to the cost of a request that is satisfied in iteration one may not be satisfied itself. Later, in iteration two, the unsatisfied data request is considered separately (because it is a different version). When considered separately, the original data item no longer has an associated cost that enables it to be satisfied in iteration two. Even if the original data item does have a cost that enables it to be satisfied, it may be satisfied later, using different time intervals on virtual links, or using different virtual links. This usage of

different network resources can then cause other data requests to be unsatisfiable using their primary version. For this reason, full—one_C1 (which does not collectively consider multiple requesting destinations) is less inclined to decrease in performance in successive iterations.

An additional reason for a lack of improvement after each iteration for data sets with high oversubscription rates is related to the large number of requests of high priority in the system. There are already very many data items in these tests with a desirable priority to select from, and the secondary versions of data items are not any better of a choice than any of the primary versions of data items that are available.

In summary, the use of multiple versions will help some heuristics improve the sum of priorities satisfied in all but the most oversubscribed cases. The improvement obtained in some operator environments exceeds 10%. In almost all cases, the best improvement is given by the second iteration of the variable time, variable accuracy algorithm.

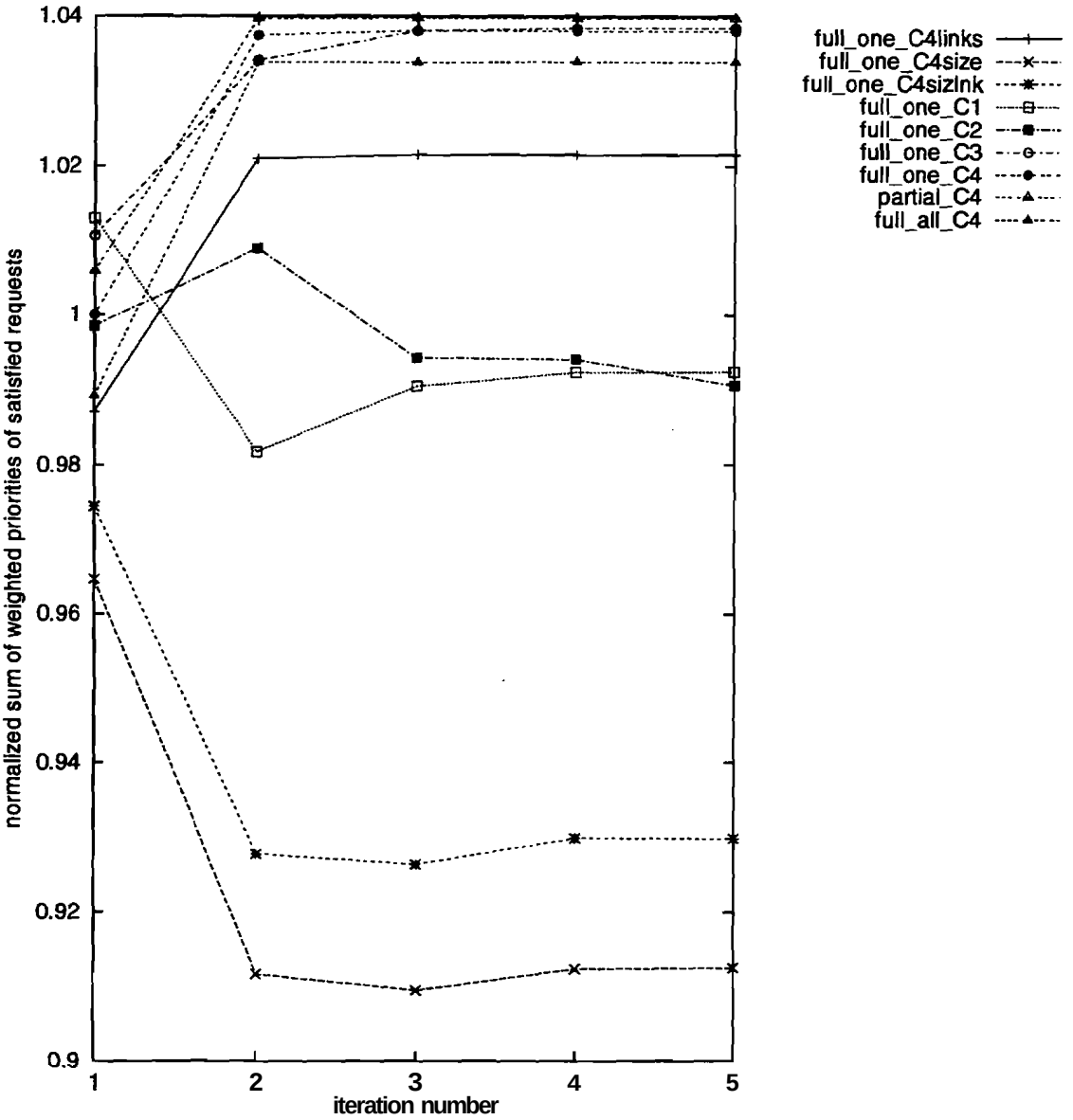

Figure 9.1: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set **had** an oversubscription rate of 0.2, an average link traversal count of 2.5, and an *w* value of 1.

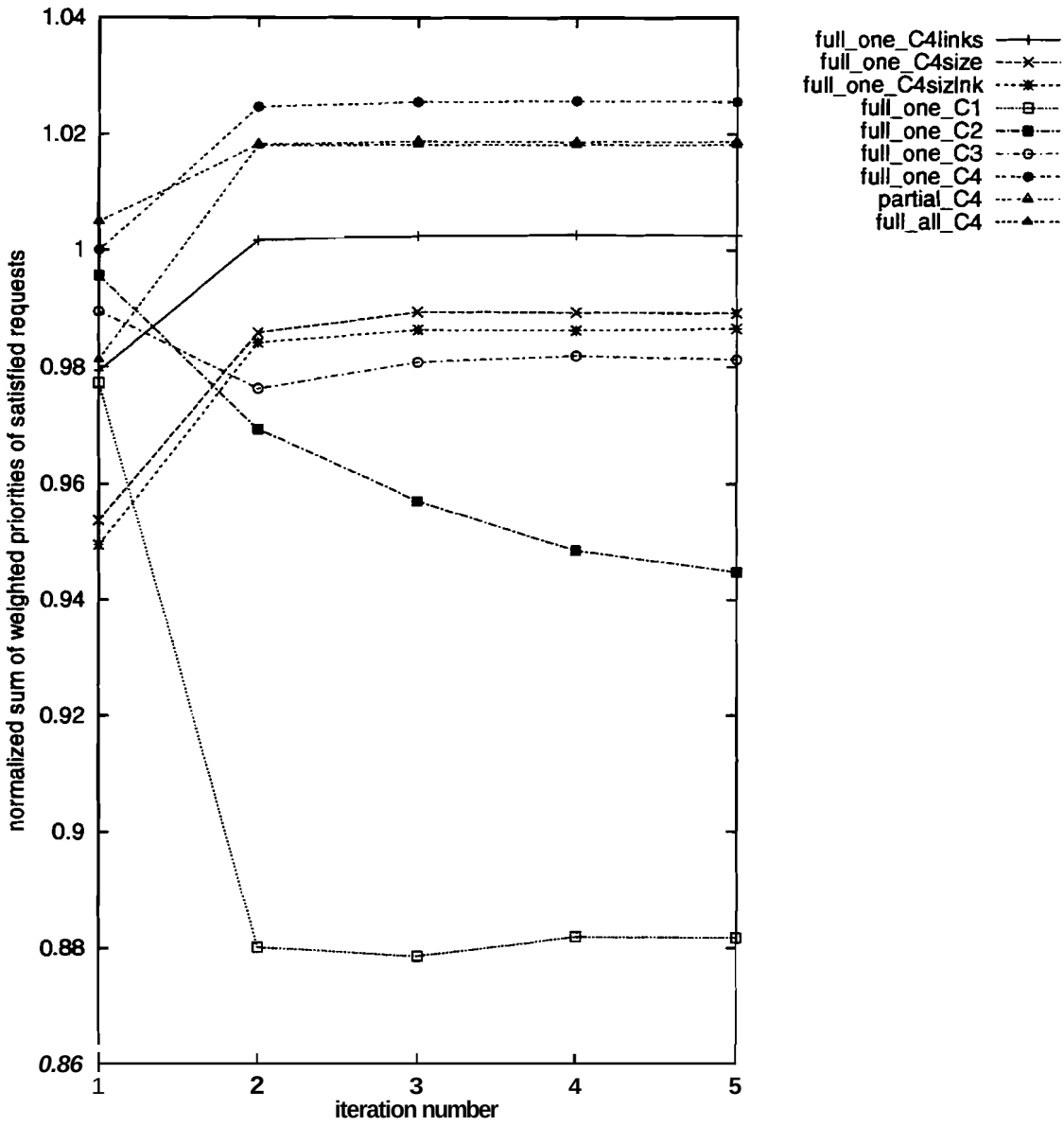

Figure 9.2: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.4, an average link traversal count of 2.5, and an w value of 1.

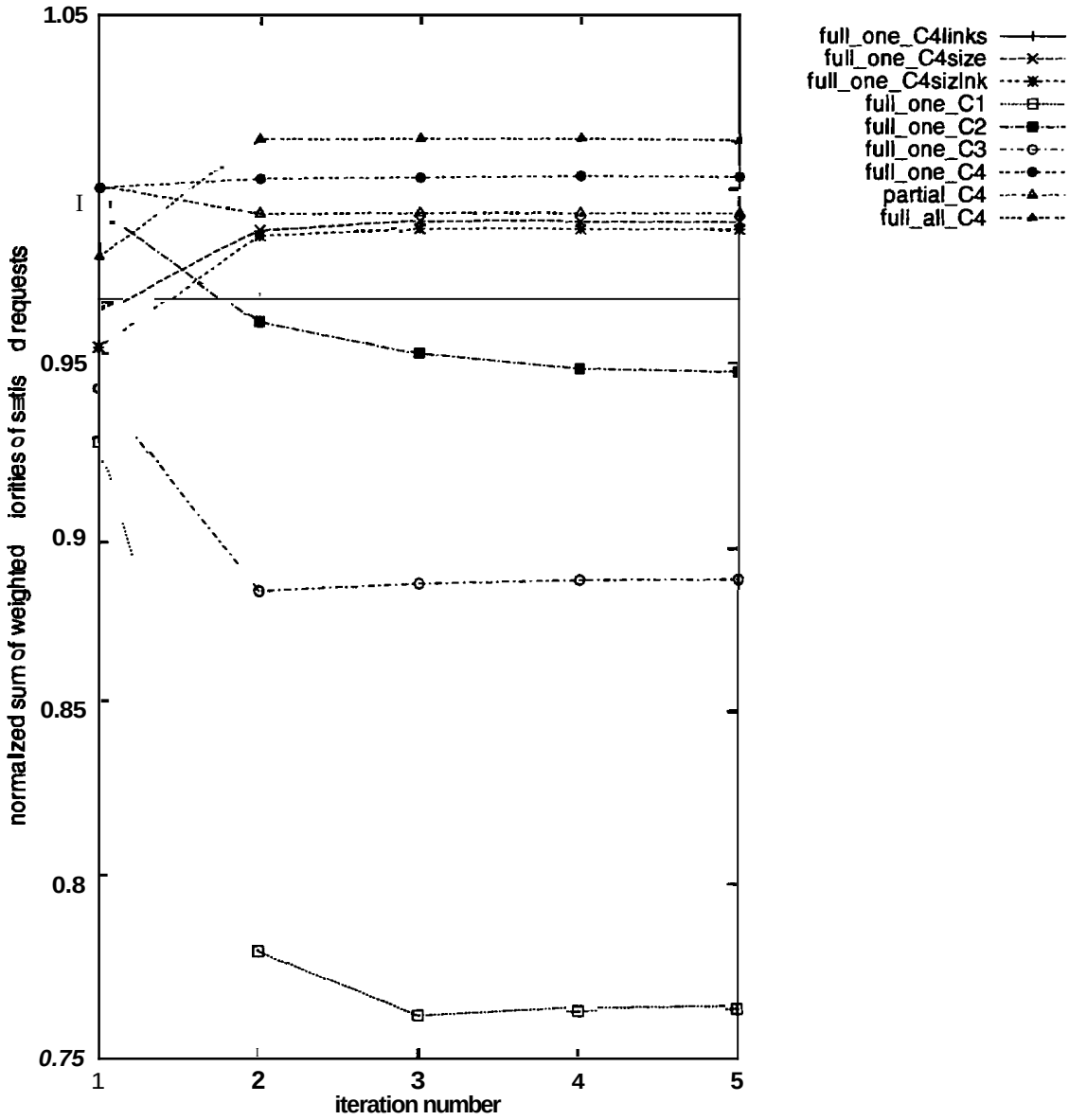

Figure 9.3: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.8, an average link traversal count of 2.5, and an ω value of 1.

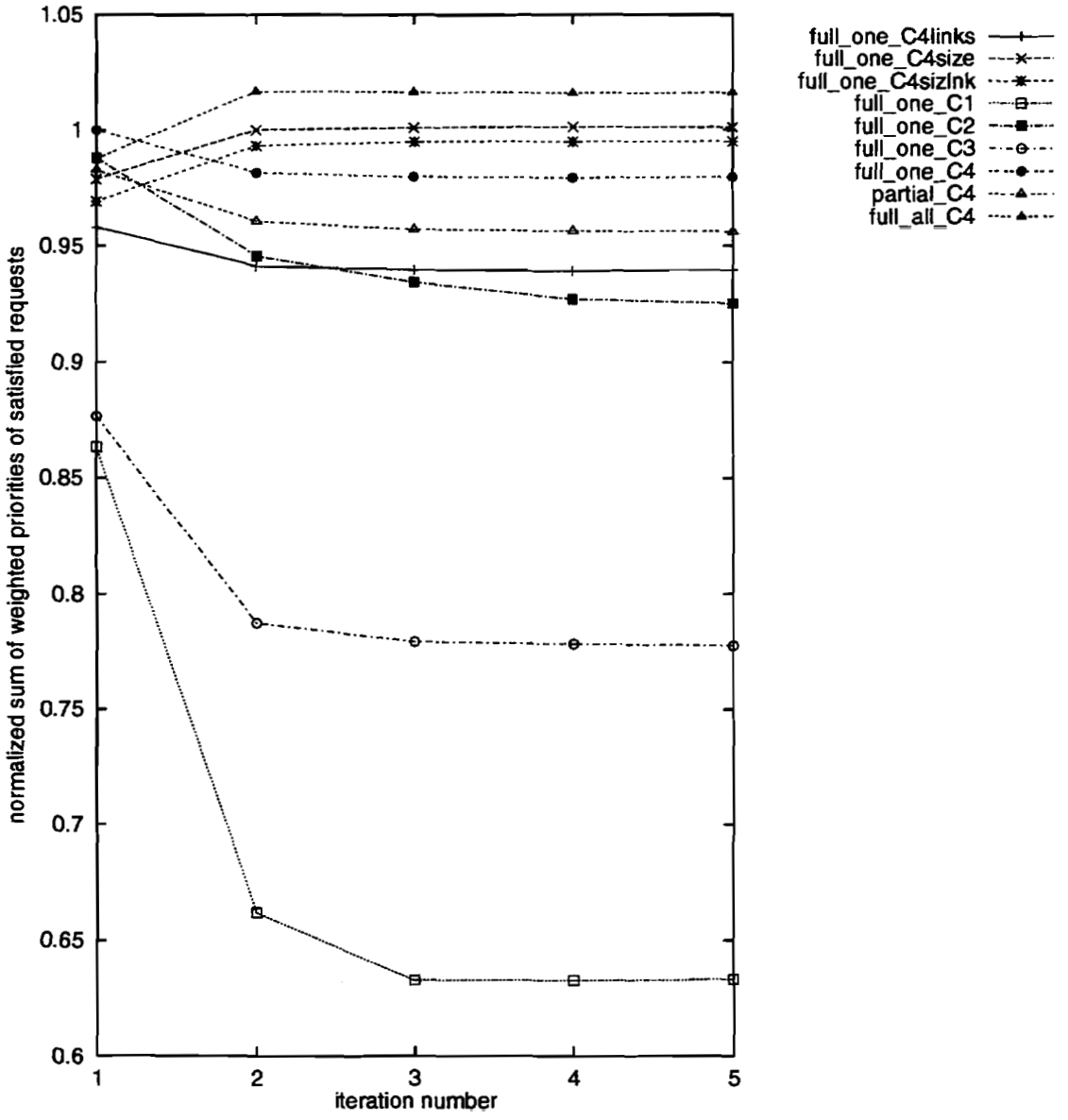

Figure 9.4: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 1.6, an average link traversal count of 2.5, and an *w* value of 1.

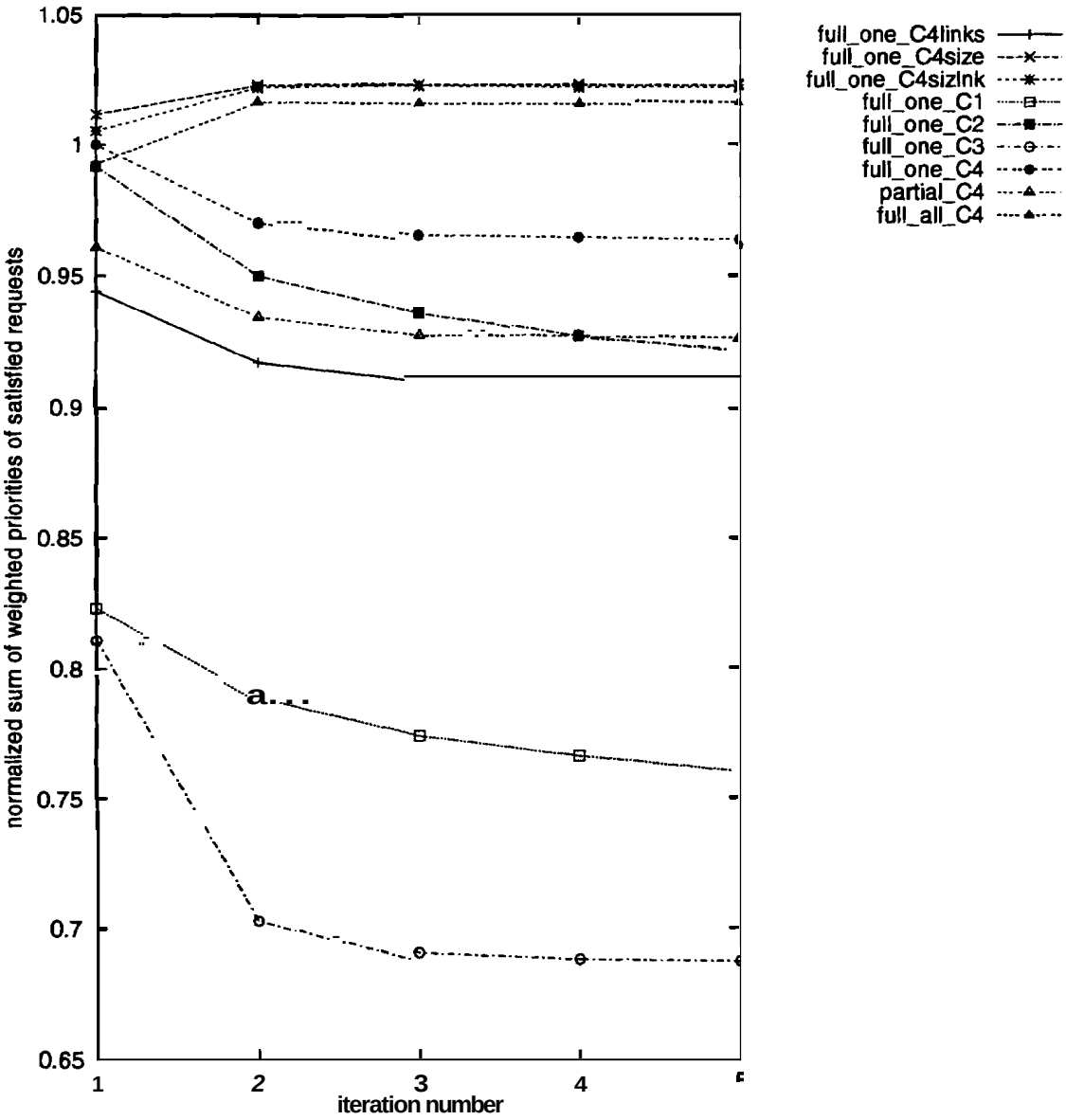

Figure 9.5: Weighted sum of satisfied requests' priorities normalized to the performance of full—one—C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 3.1, an average link traversal count of 2.5, and an w value of 1.

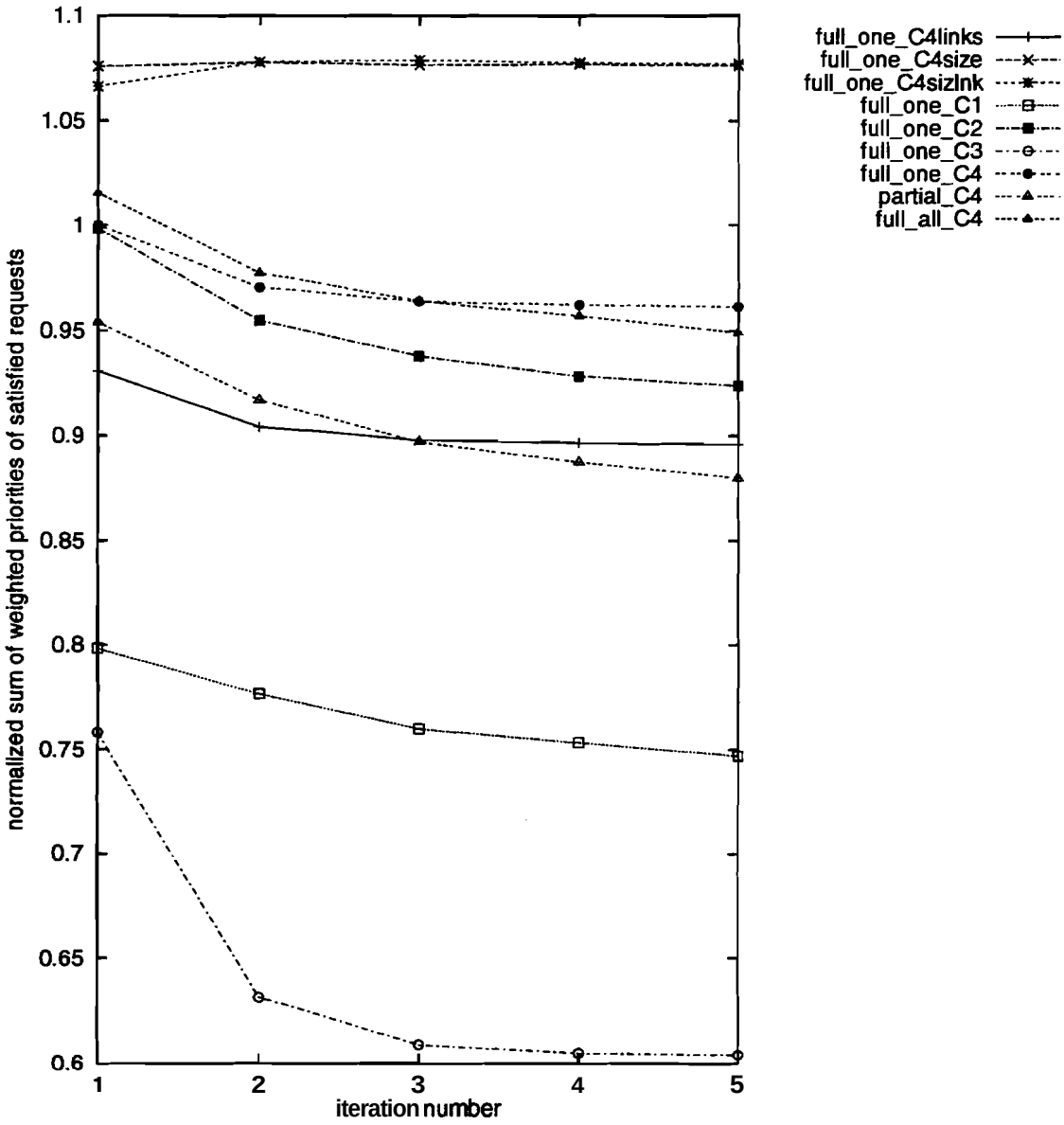

Figure **9.6:** Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of **6.2,** an average link traversal count of 2.5, and an w value of 1.

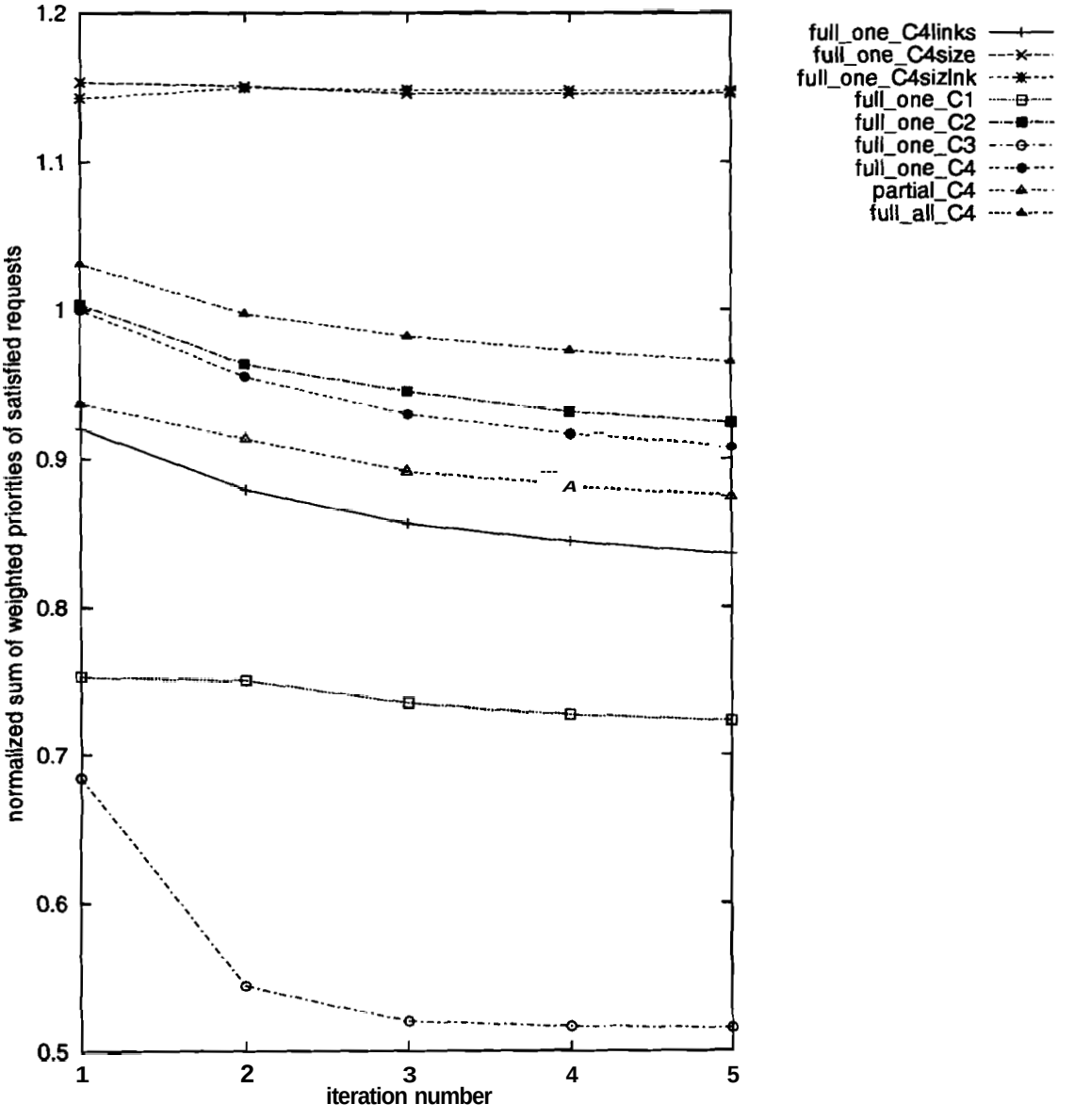

Figure 9.7: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 12.5, an average link traversal count of 2.5, and an *w* value of 1.

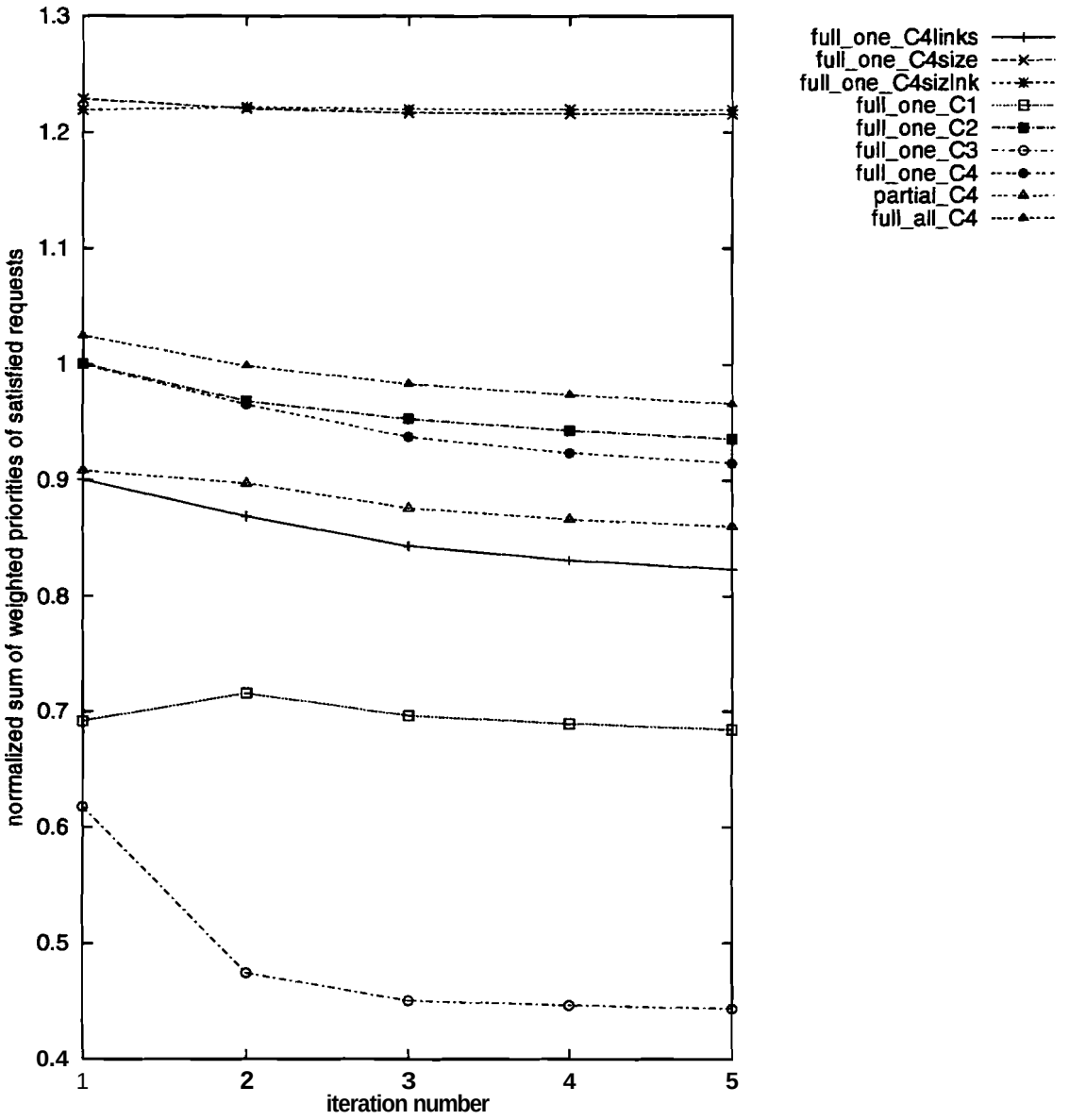

Figure 9.8: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 25.0, an average link traversal count of 2.5, and an ω value of 1.

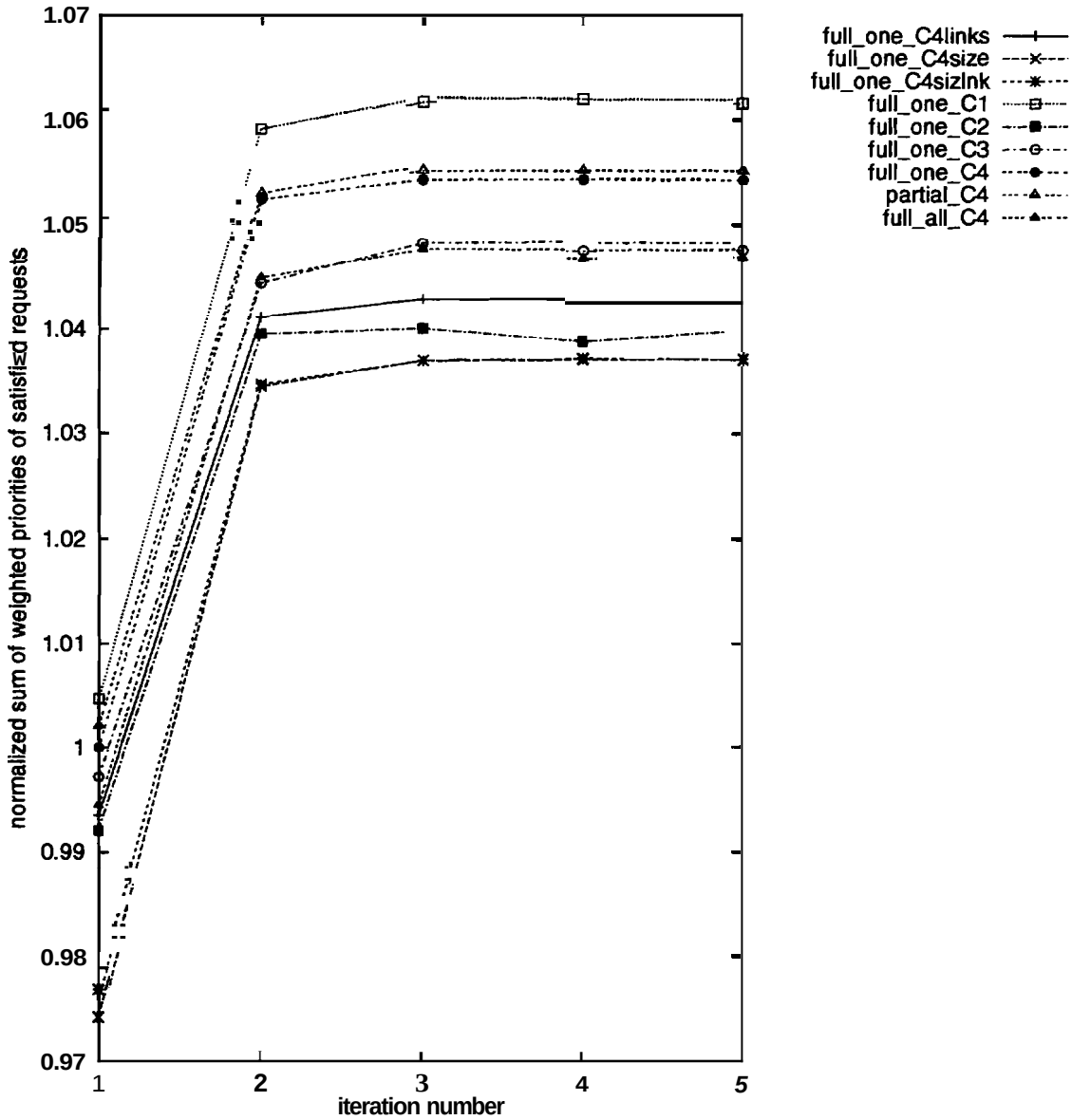

Figure 9.9: Weighted sum of satisfied requests' priorities normalized to the performance of full_one_C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.2, an average link traversal count of 2.5, and an *w* value of 2.

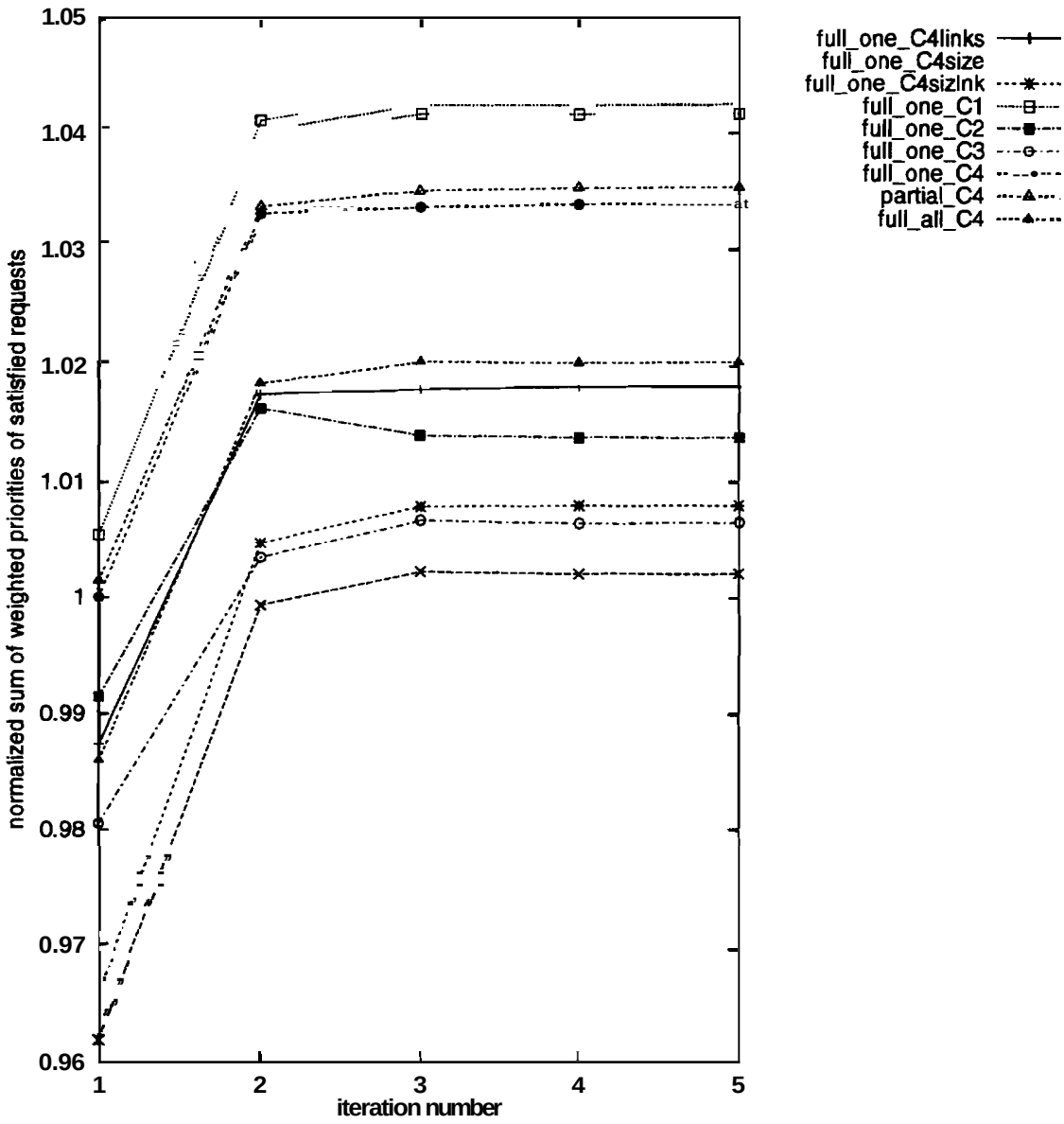

Figure 9.10: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.4, an average link traversal count of 2.5, and an w value of 2.

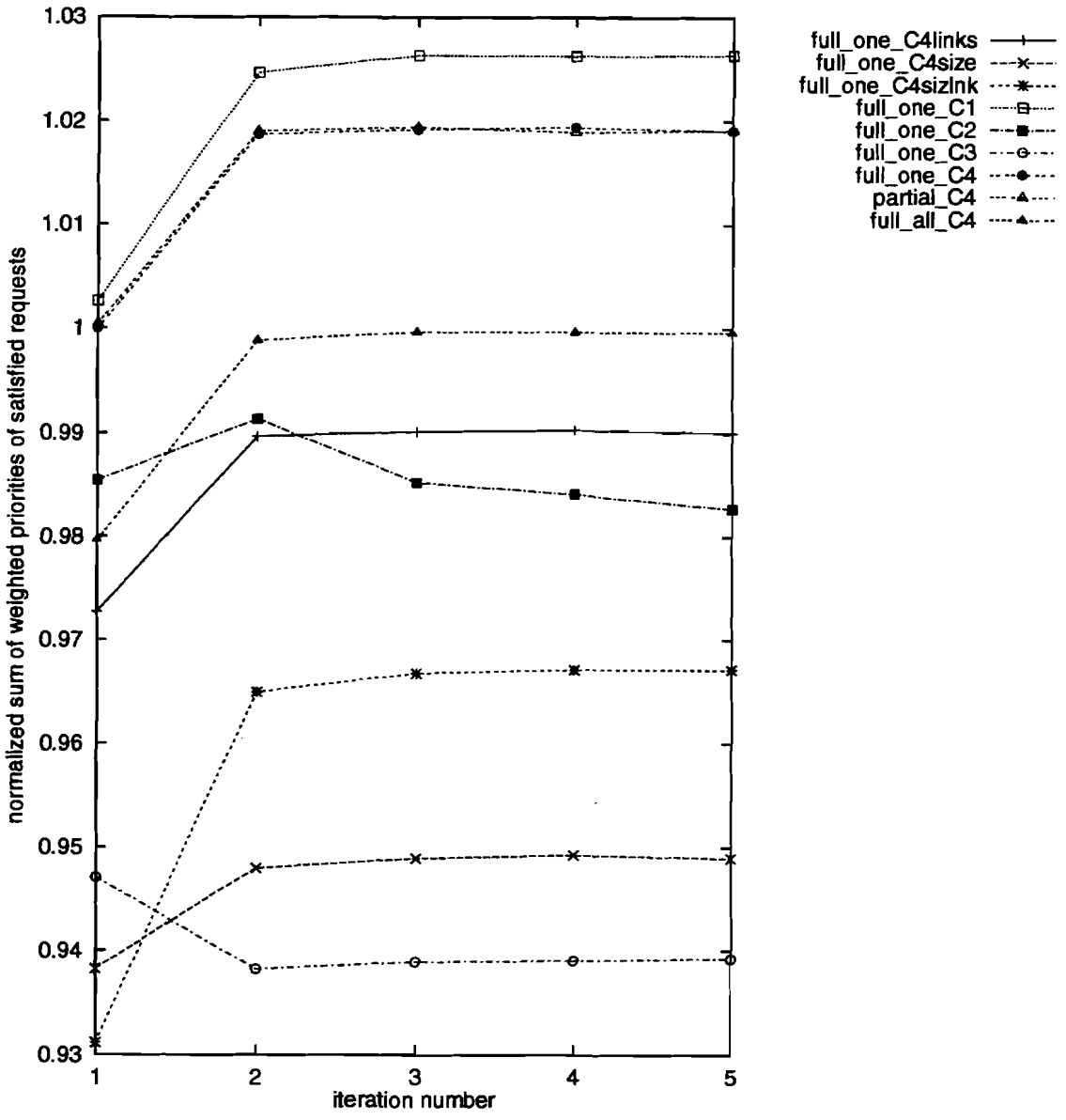

Figure 9.11: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.8, an average link traversal count of 2.5, and an w value of 2.

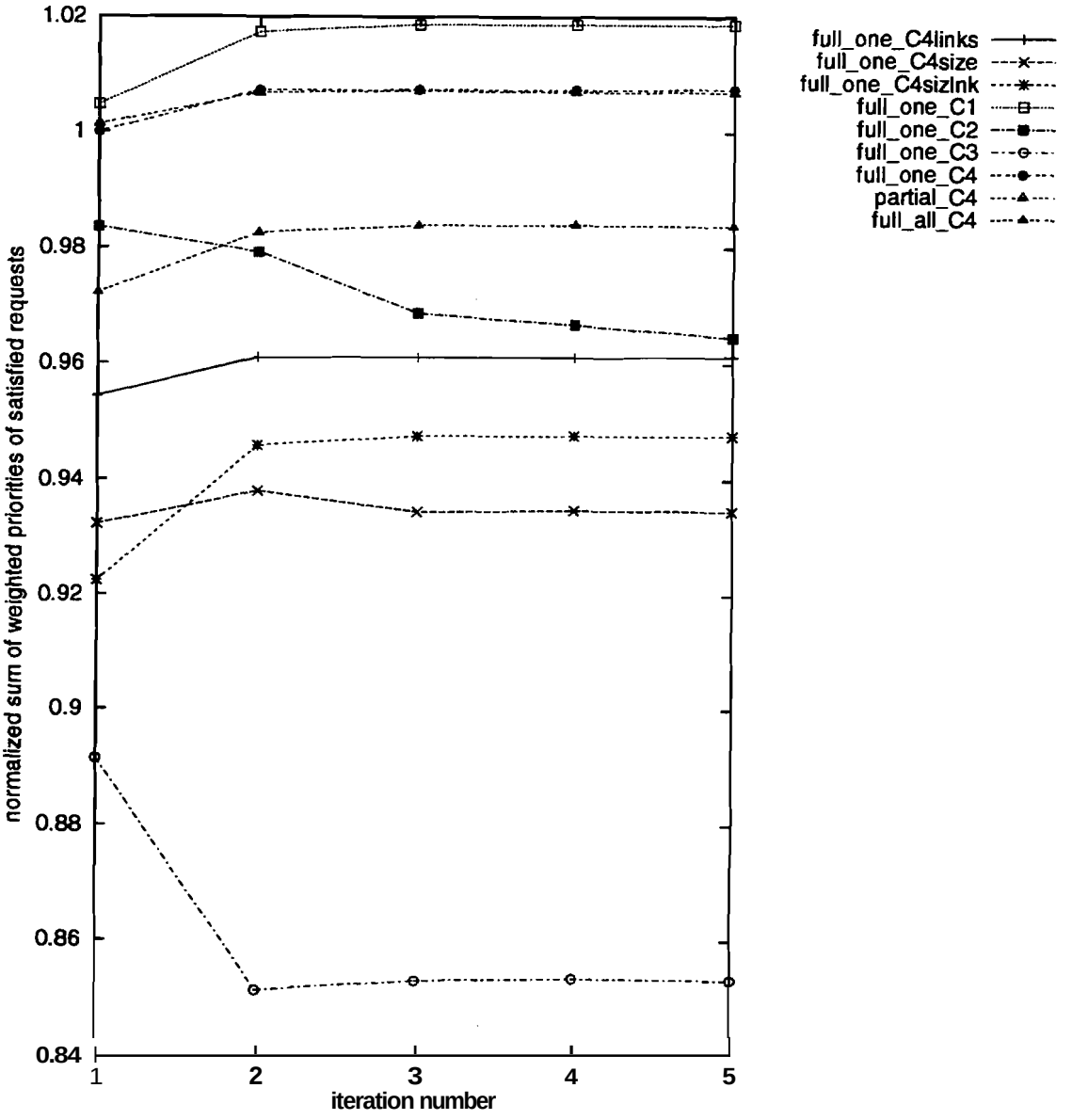

Figure 9.12: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 1.6, an average link traversal count of 2.5, and an w value of 2.

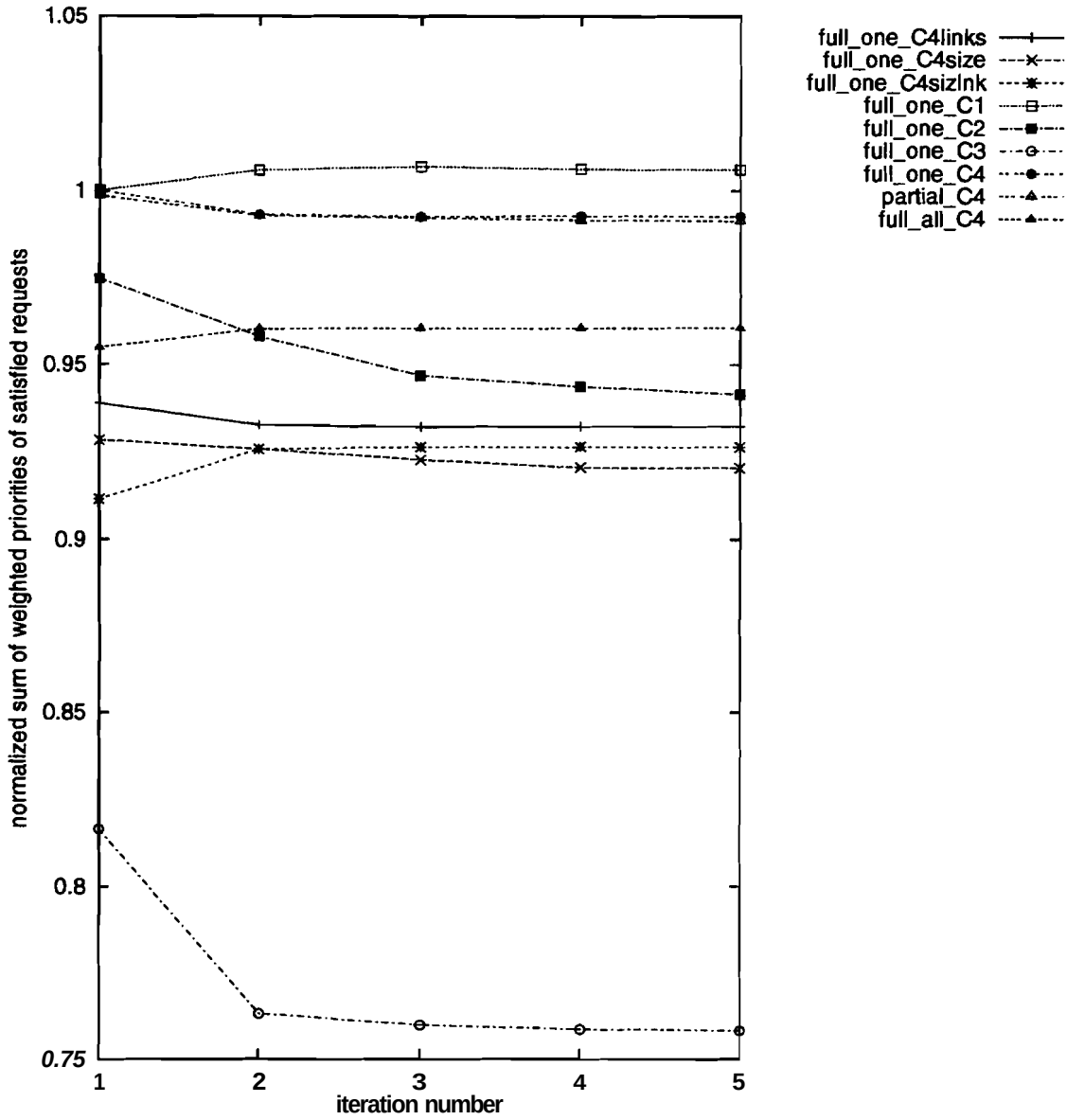

Figure 9.13: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 3.1, an average link traversal count of 2.5, and an w value of 2.

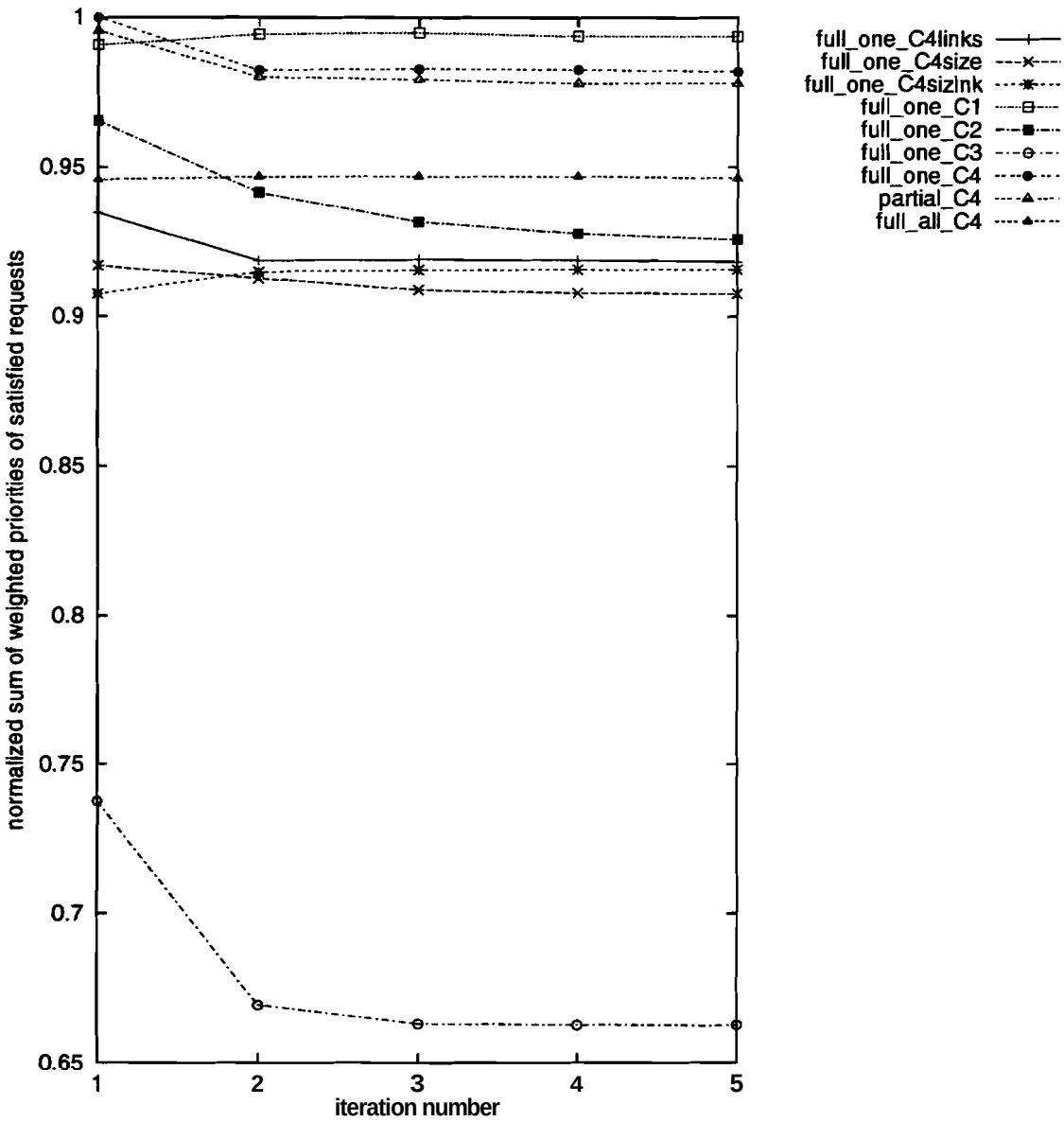

Figure 9.14: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 6.2, an average link traversal count of 2.5, and an ω value of 2.

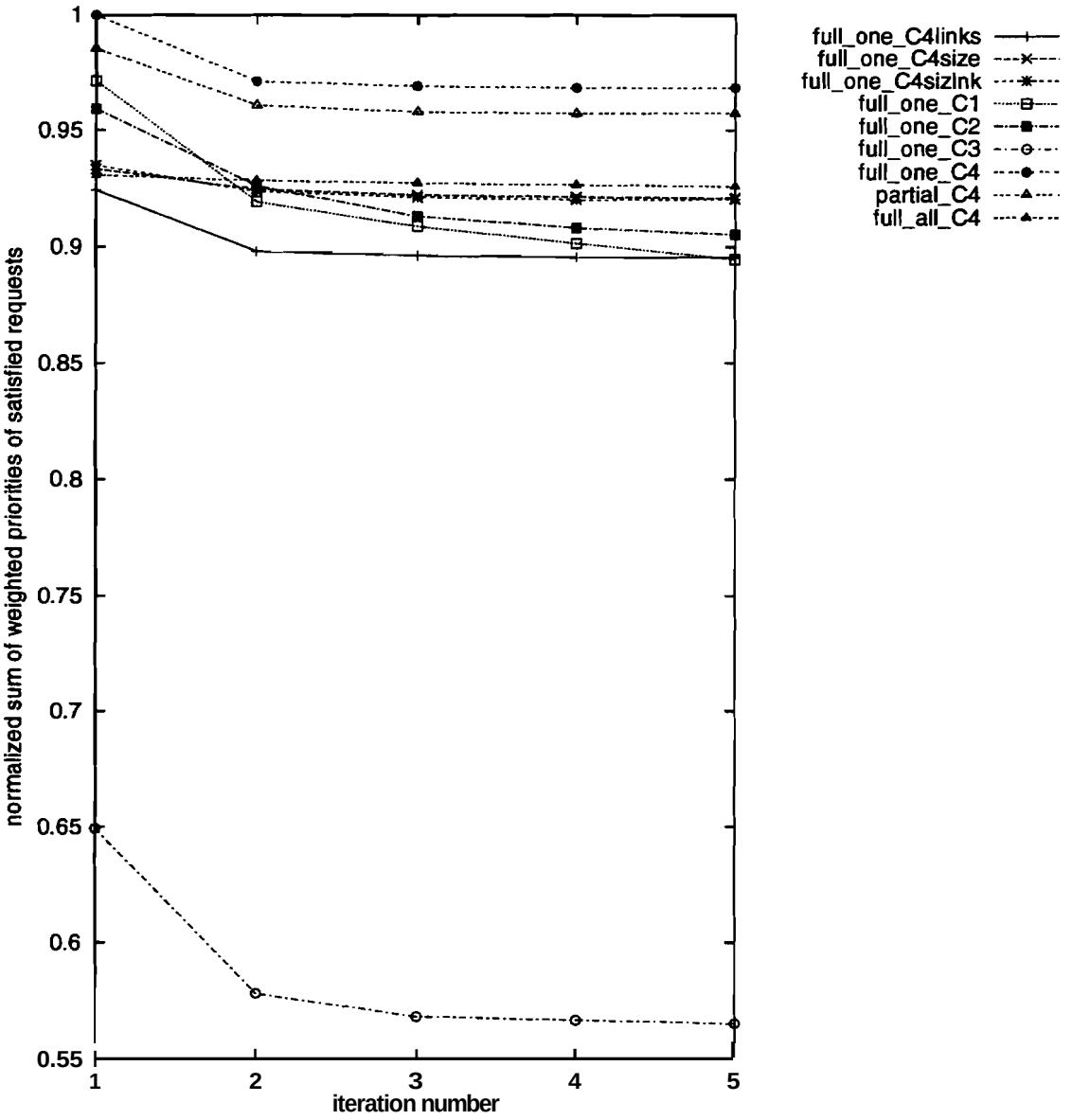

Figure 9.15: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 12.5, an average link traversal count of 2.5, and an ω value of 2.

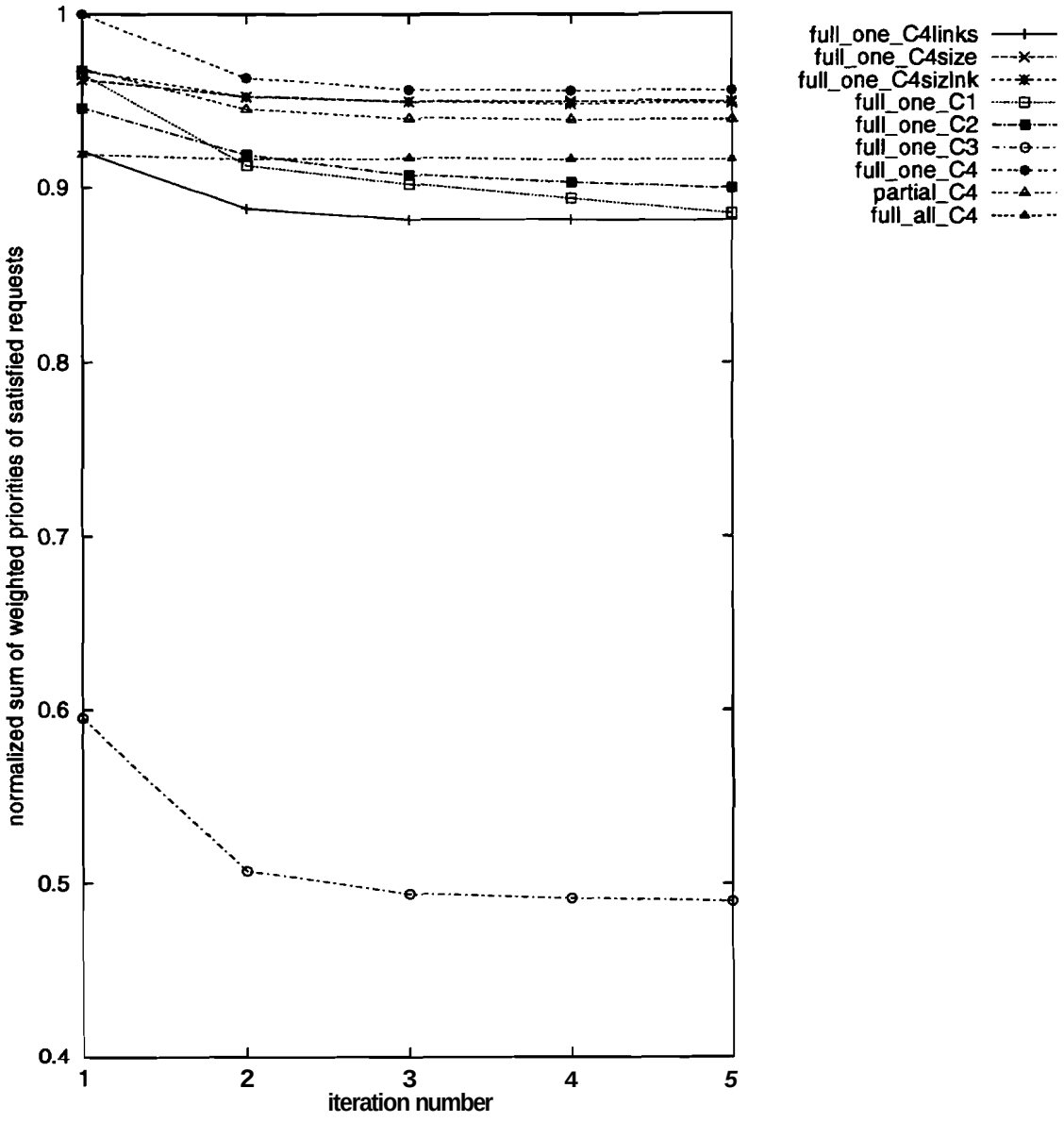

Figure 9.16: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 25.0, an average link traversal count of 2.5, and an ω value of 2.

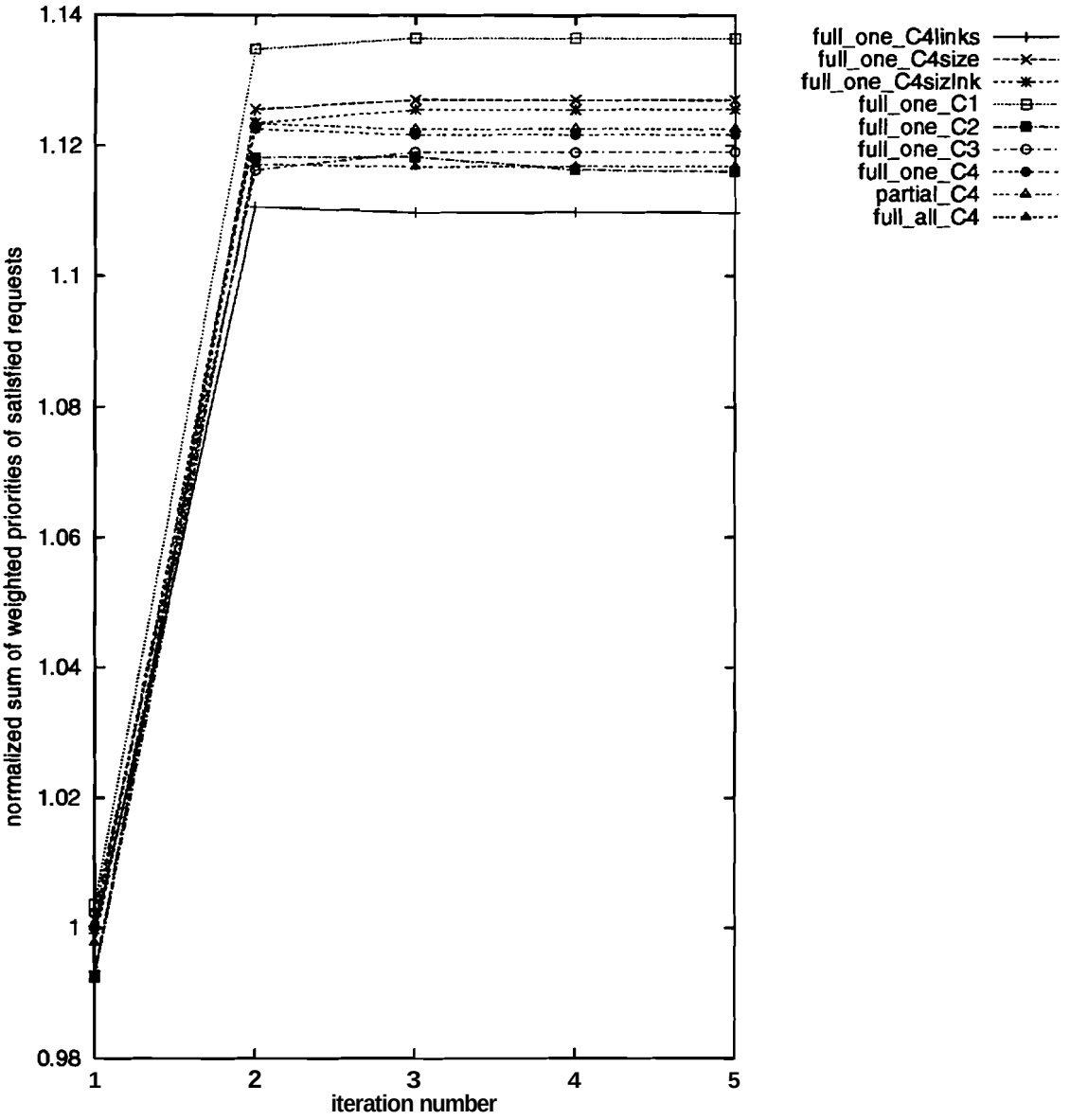

Figure 9.17: Weighted sum of satisfied requests' priorities normalized to the performance of full one–C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.2, an average link traversal count of 2.5, and an *w* value of 4.

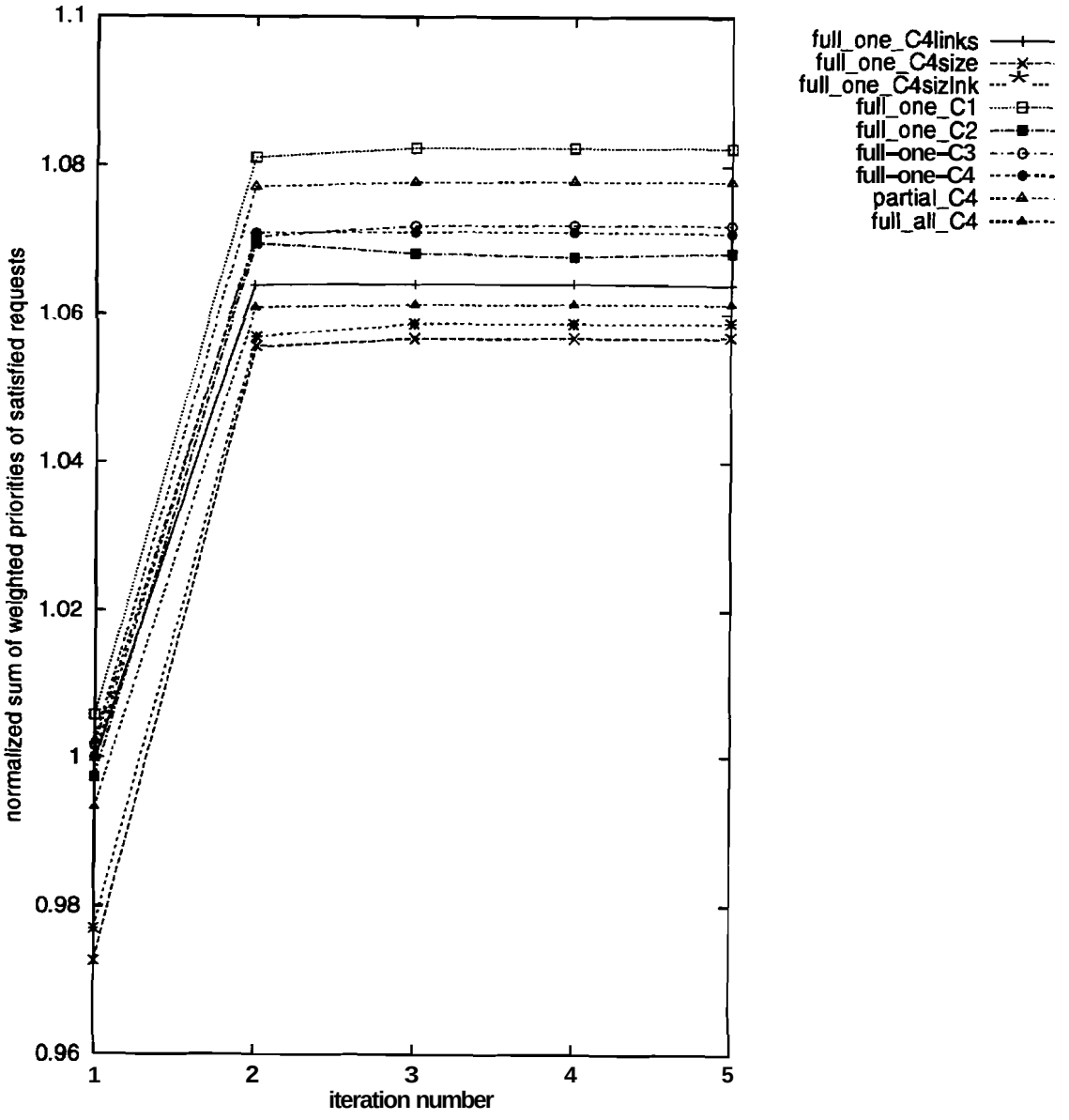

Figure 9.18: Weighted sum of satisfied requests' priorities normalized to the performance of full—one—C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.4, an average link traversal count of 2.5, and an w value of 4.

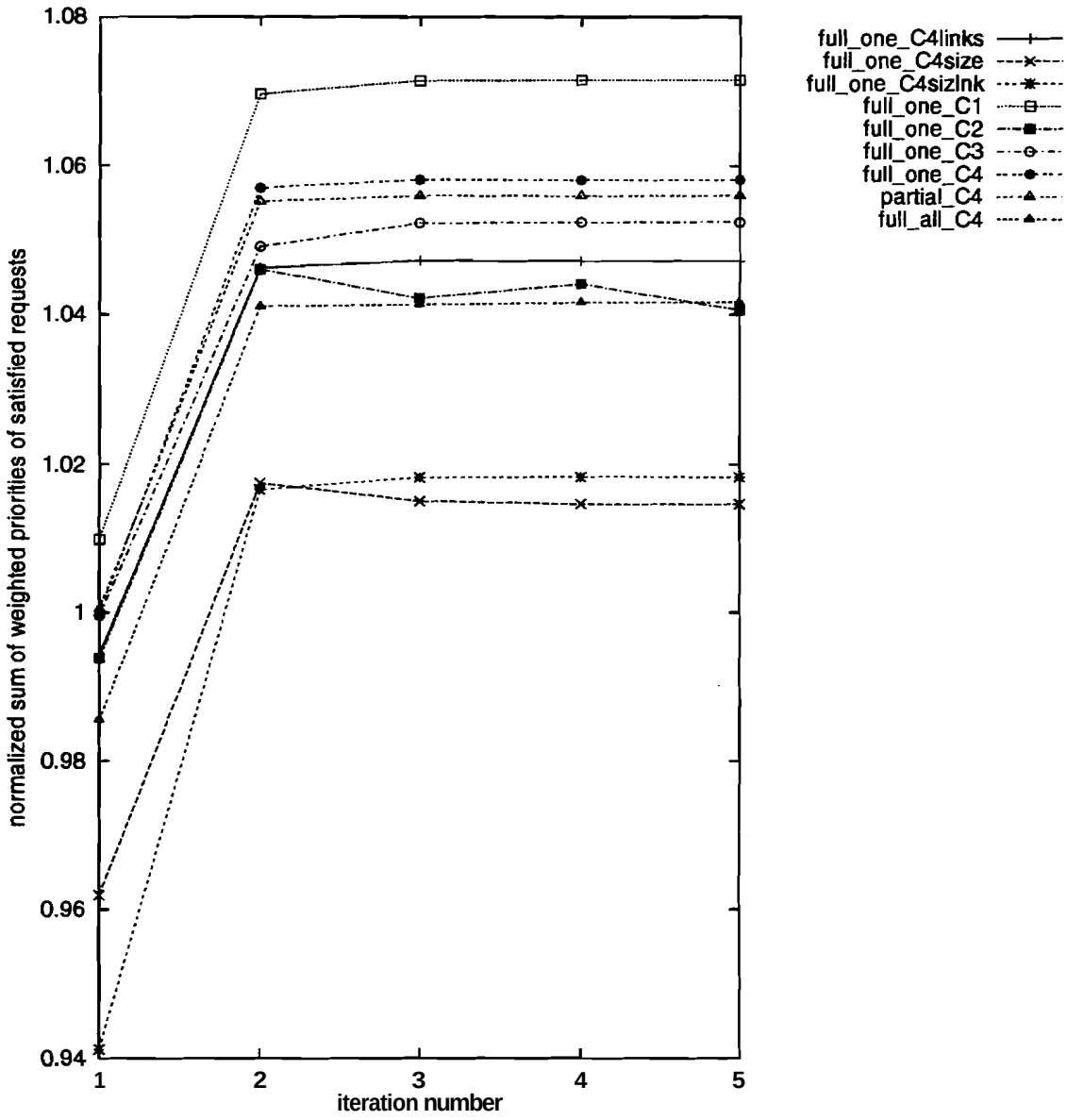

Figure 9.19: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.8, an average link traversal count of 2.5, and an *w* value of 4.

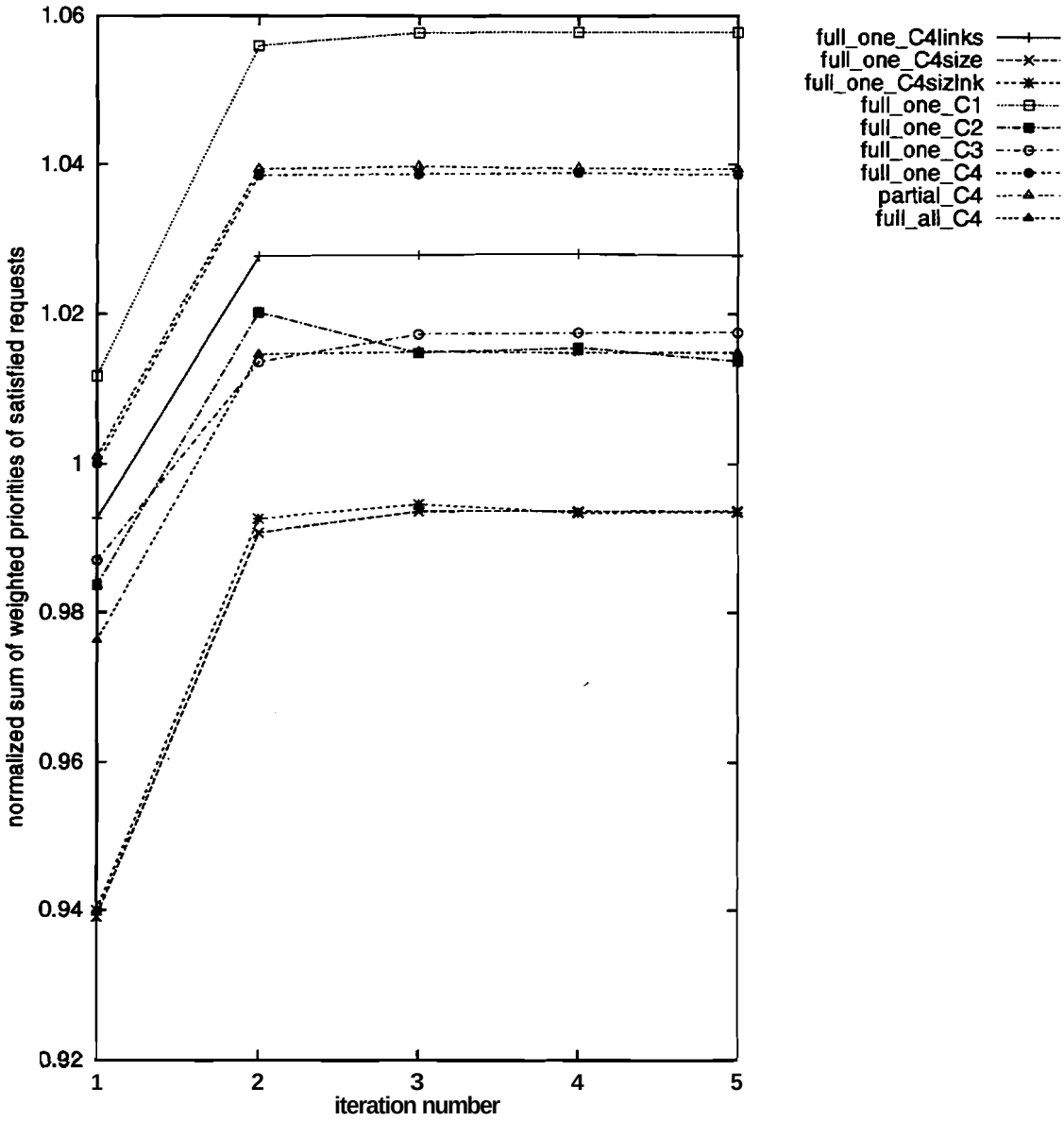

Figure 9.20: Weighted sum of satisfied requests' priorities normalized to the performance of full—one—C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 1.6, an average link traversal count of 2.5, and an w value of 4.

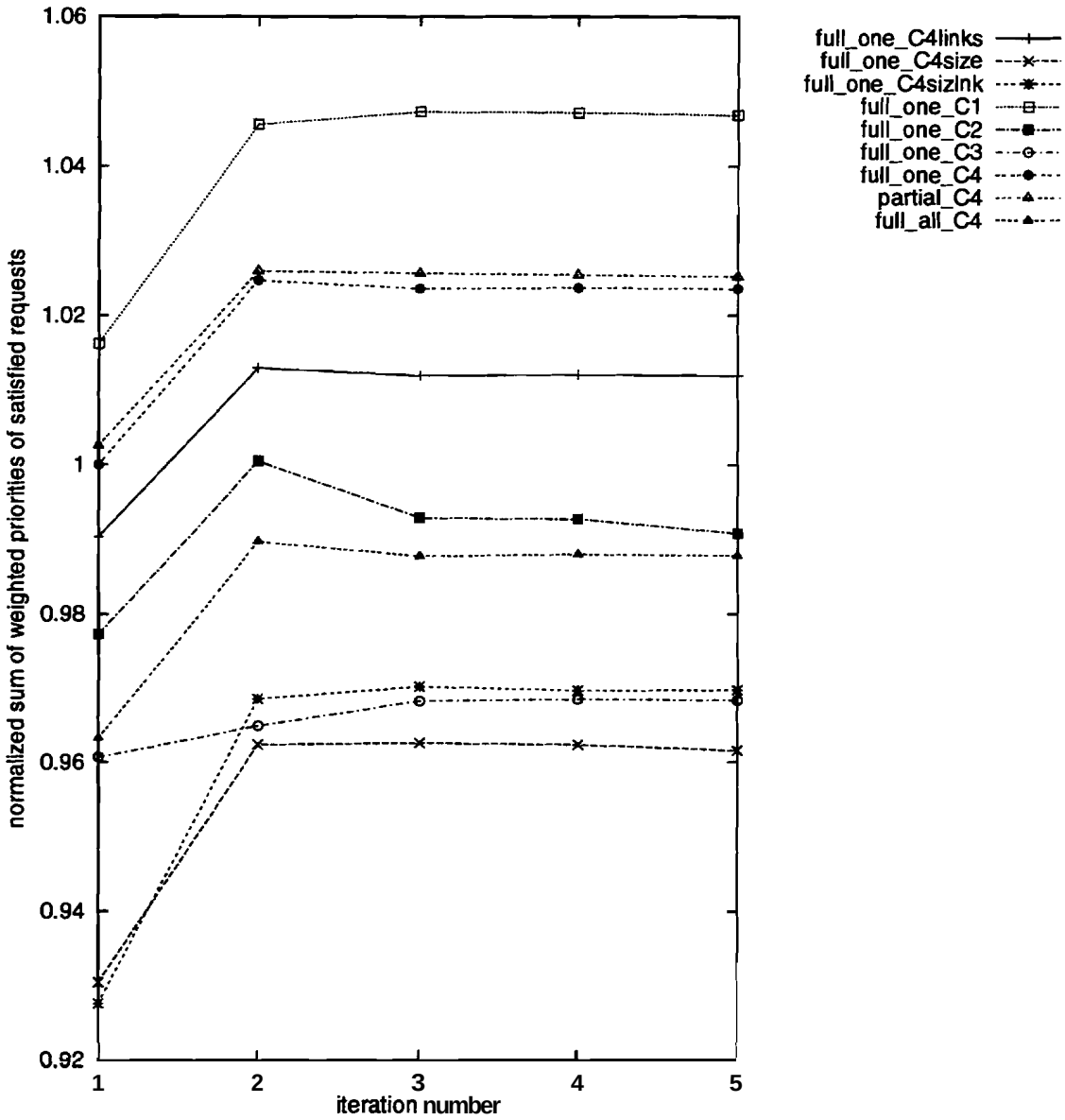

Figure 9.21: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 3.1, an average link traversal count of 2.5, and an ω value of 4.

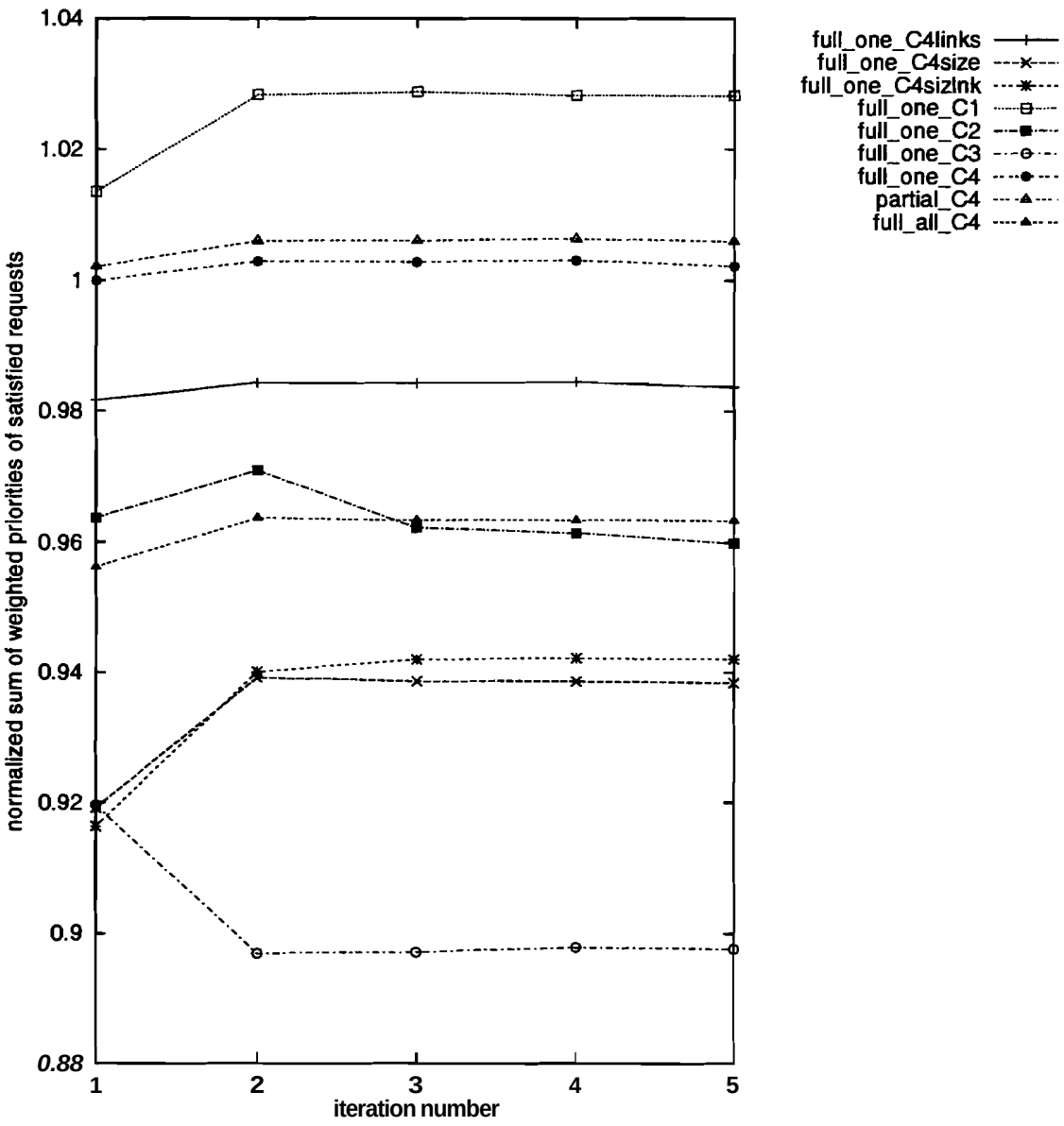

Figure 9.22: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 6.2, an average link traversal count of 2.5, and an w value of 4.

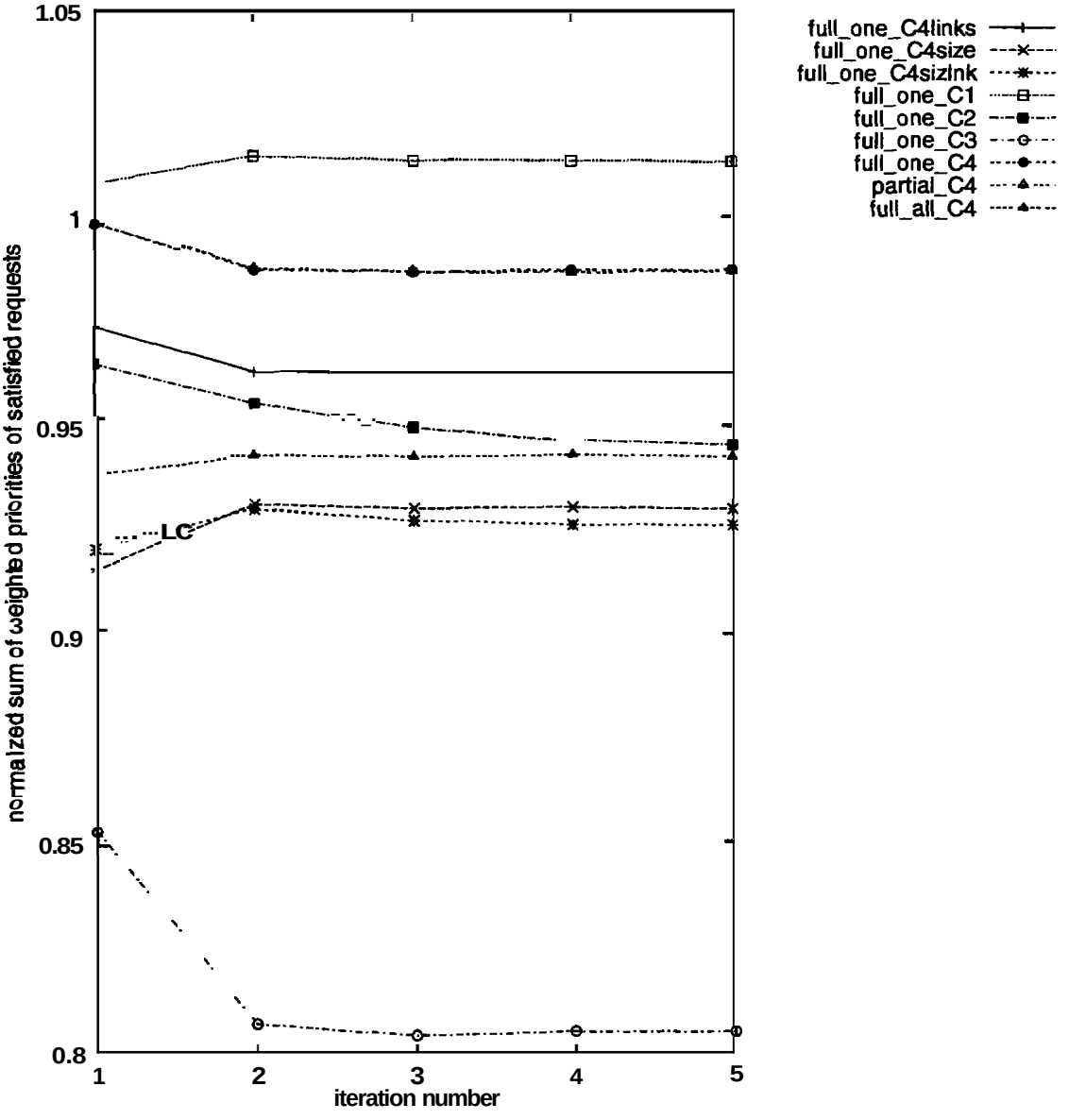

Figure **9.23:** Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set **had** an oversubscription rate of 12.5, an average link traversal count of **2.5,** and an ω value of 4.

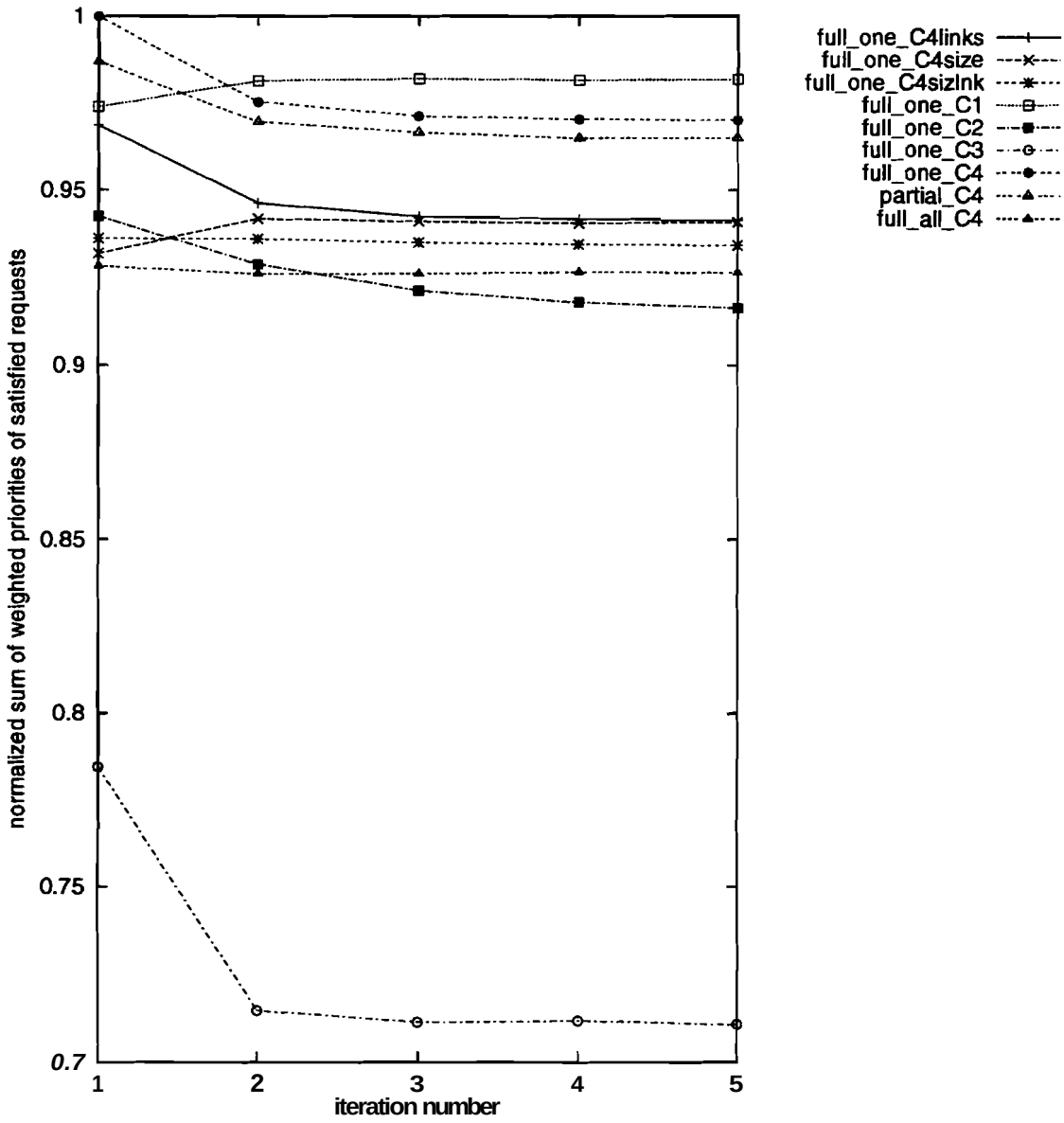

Figure 9.24: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 25.0, an average link traversal count of 2.5, and an *w* value of 4.

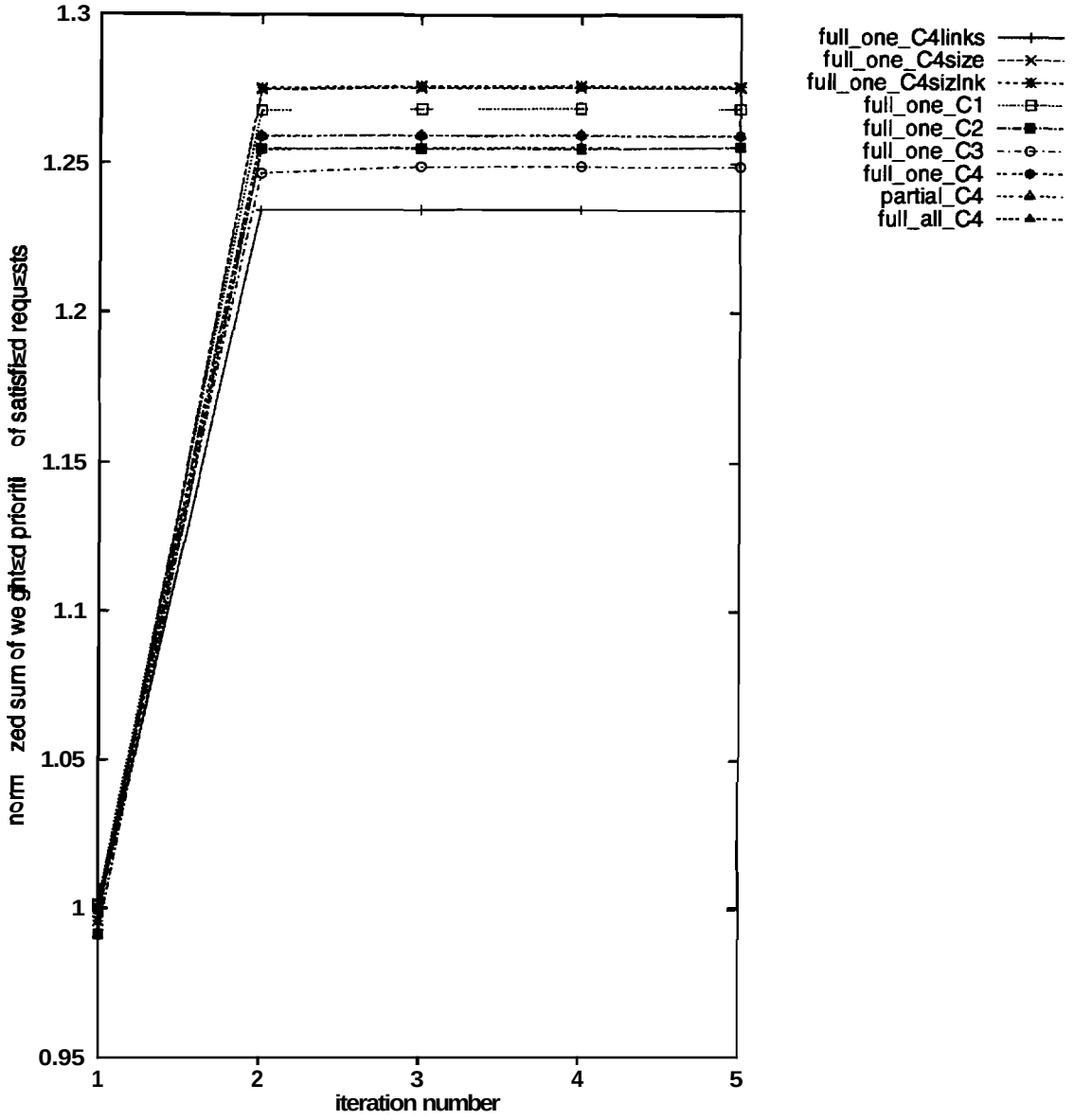

Figure 9.25: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.2, an average link traversal count of 2.5, and an w value of 8.

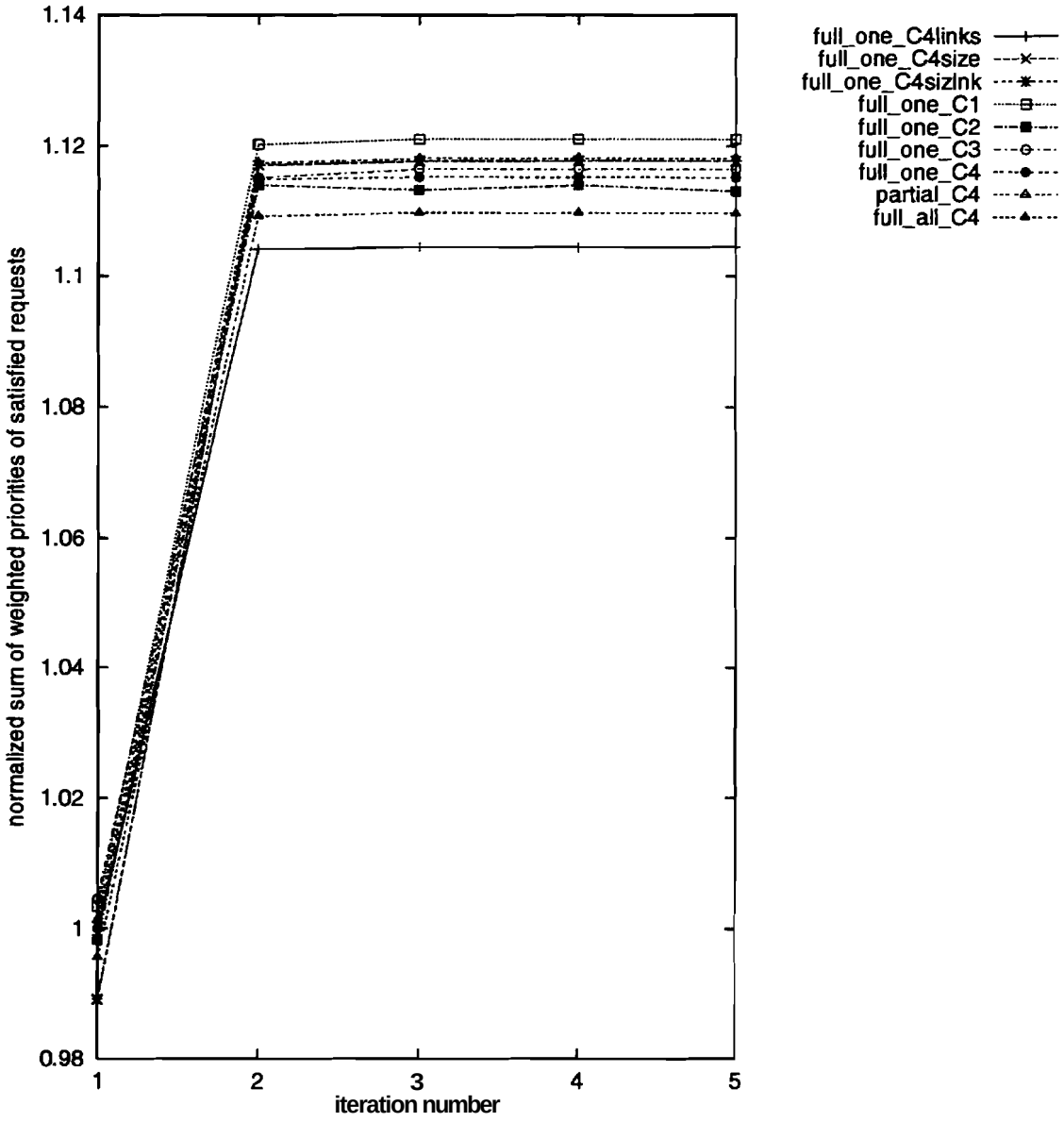

Figure 9.26: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.4, an average link traversal count of 2.5, and an w value of 8.

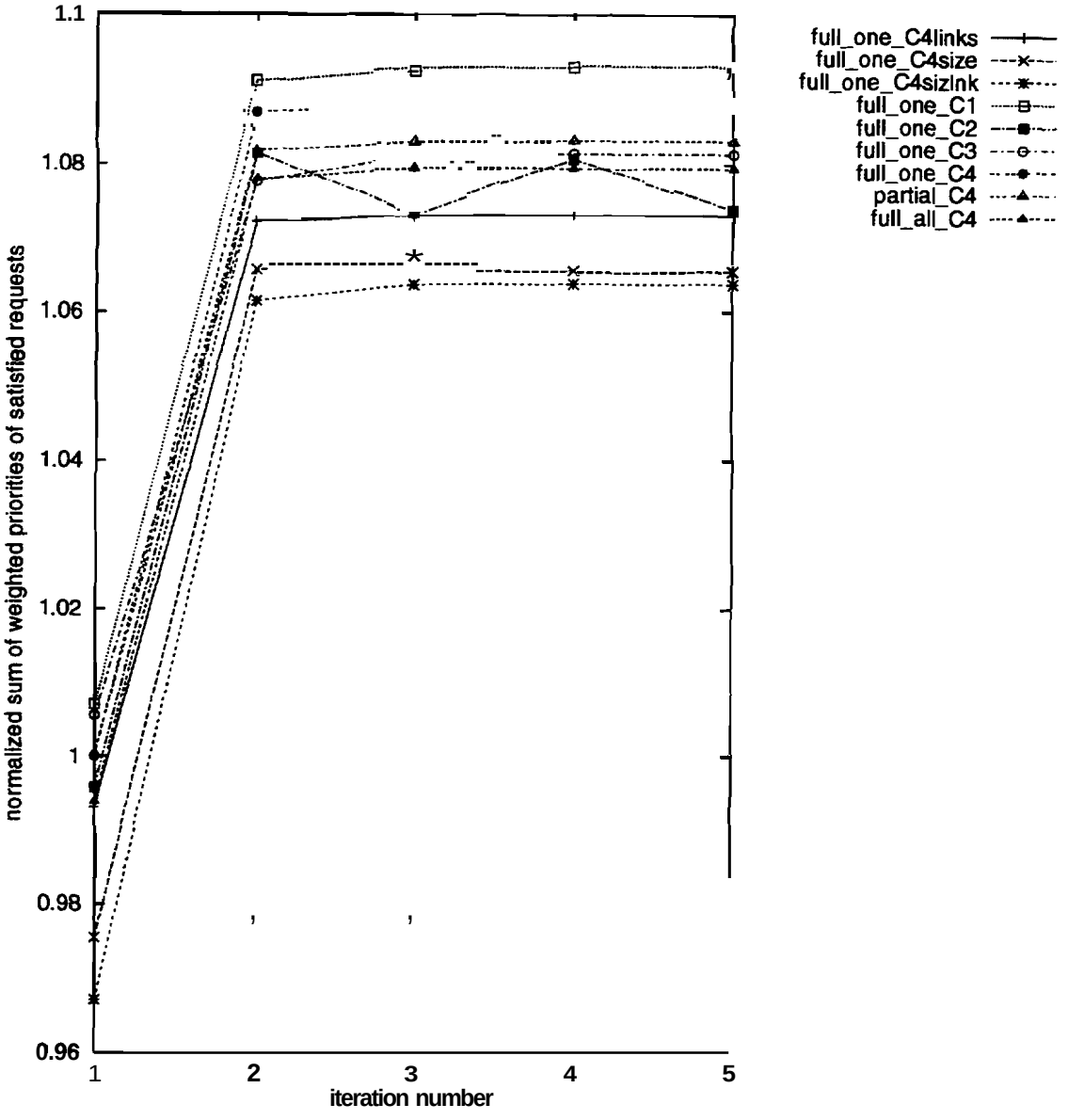

Figure 9.27: Weighted sum of satisfied requests' priorities normalized to the performance of full one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.8, an average link traversal count of 2.5, and an ω value of 8.

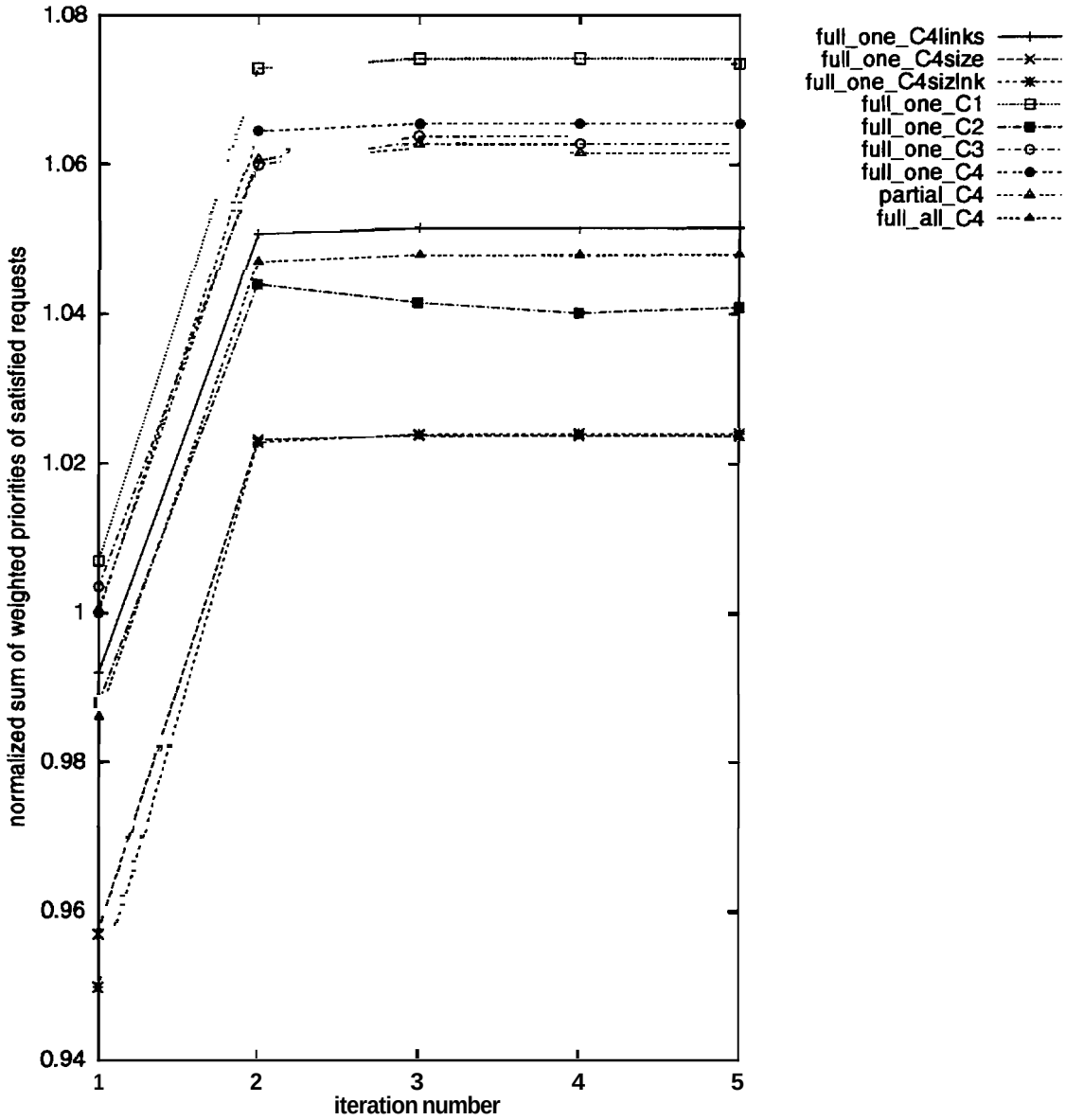

Figure 9.28: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 1.6, an average link traversal count of 2.5, and an *w* value of 8.

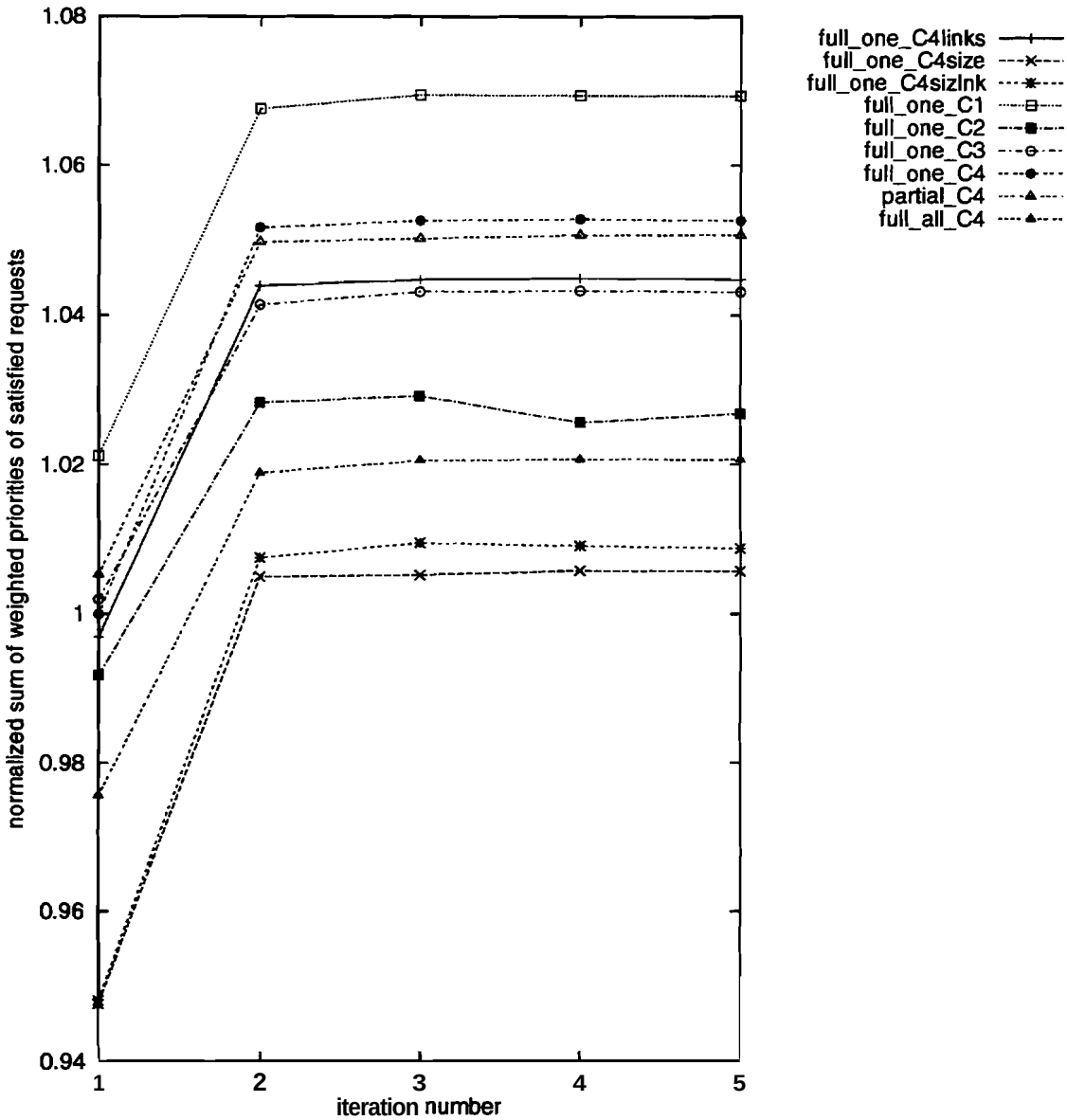

Figure 9.29: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 3.1, an average link traversal count of 2.5, and an *w* value of 8.

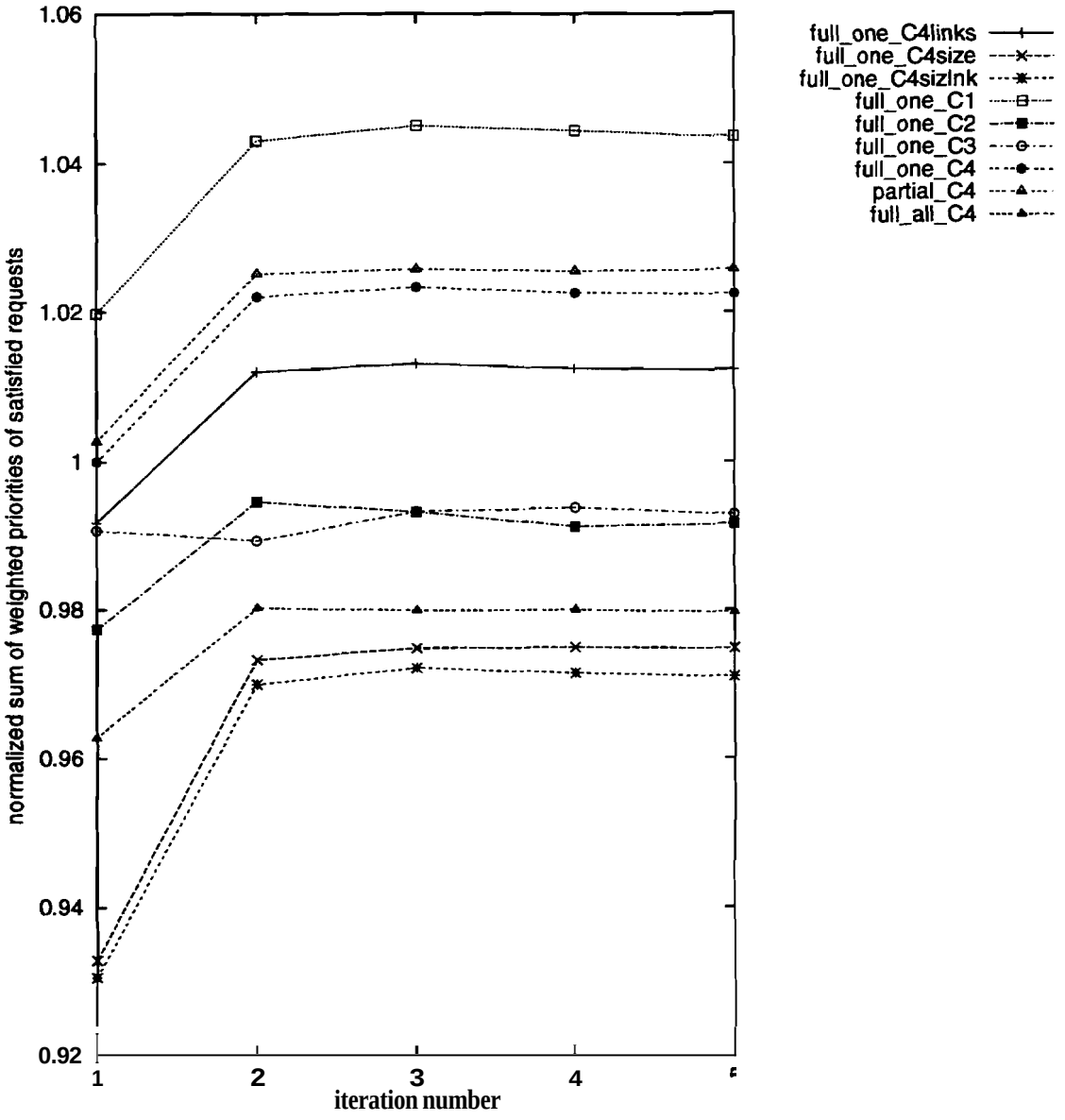

Figure 9.30: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 6.2, an average link traversal count of 2.5, and an w value of 8.

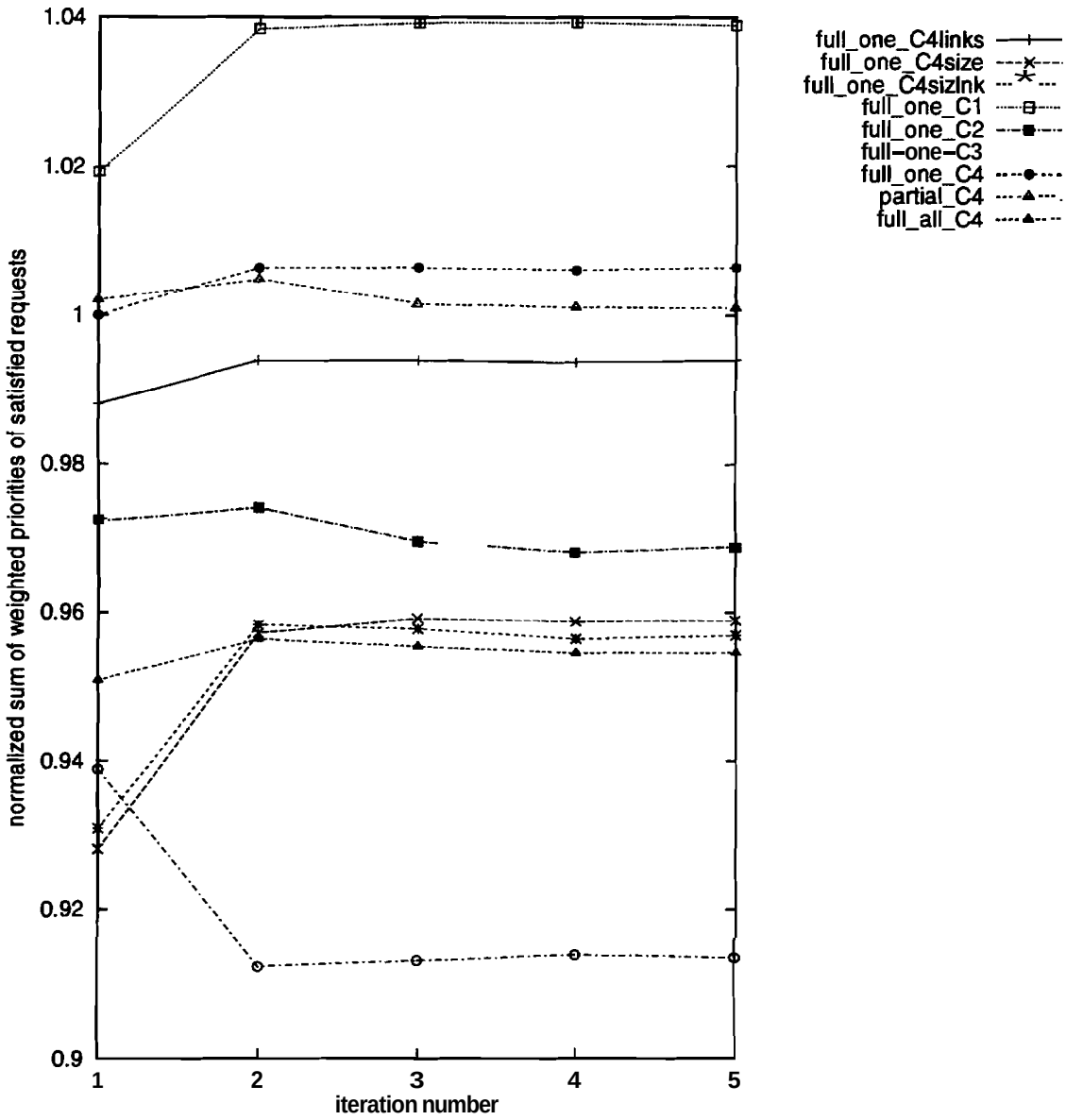

Figure 9.31: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 12.5, an average link traversal count of 2.5, and an w value of 8.

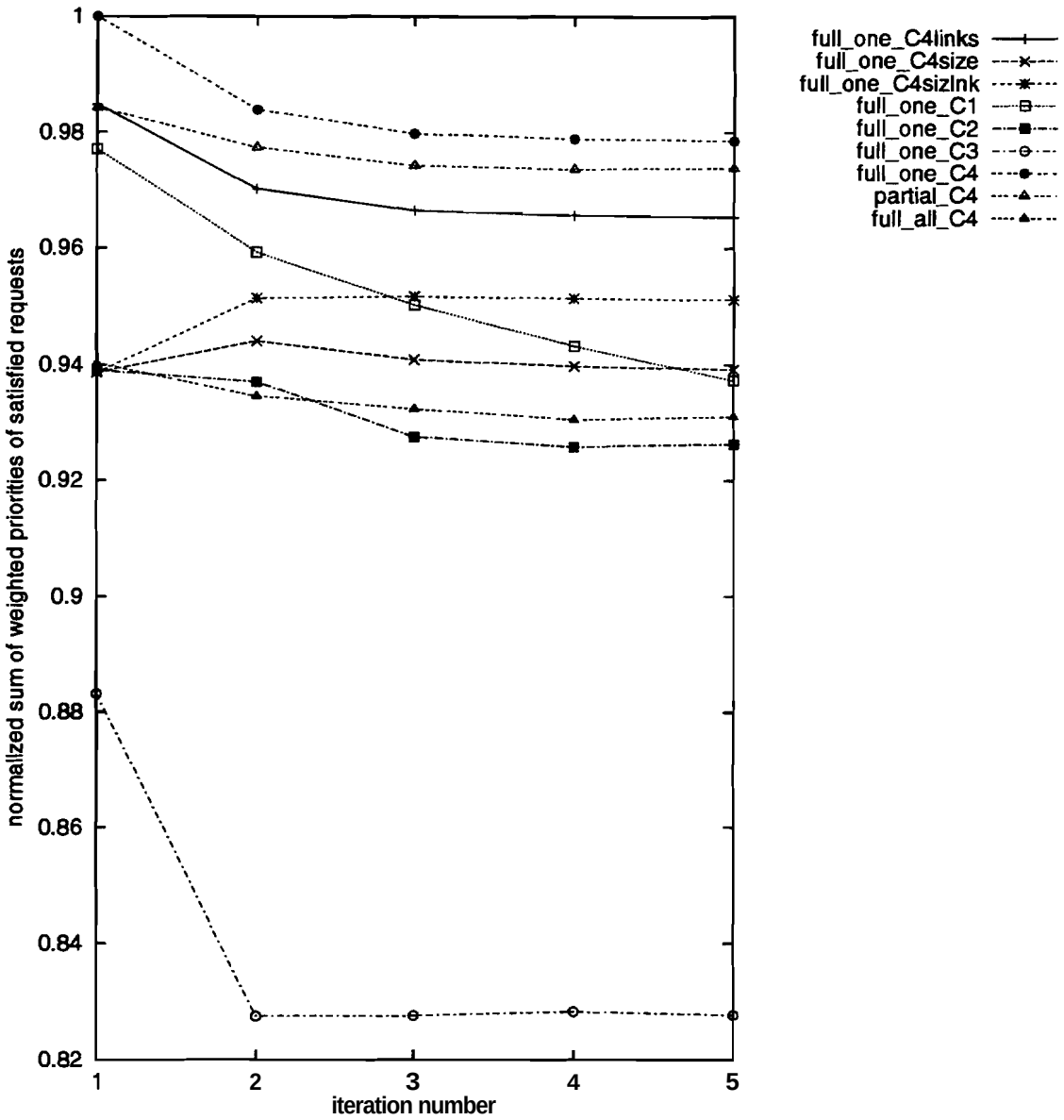

Figure 9.32: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 25.0, an average link traversal count of 2.5, and an ω value of 8.

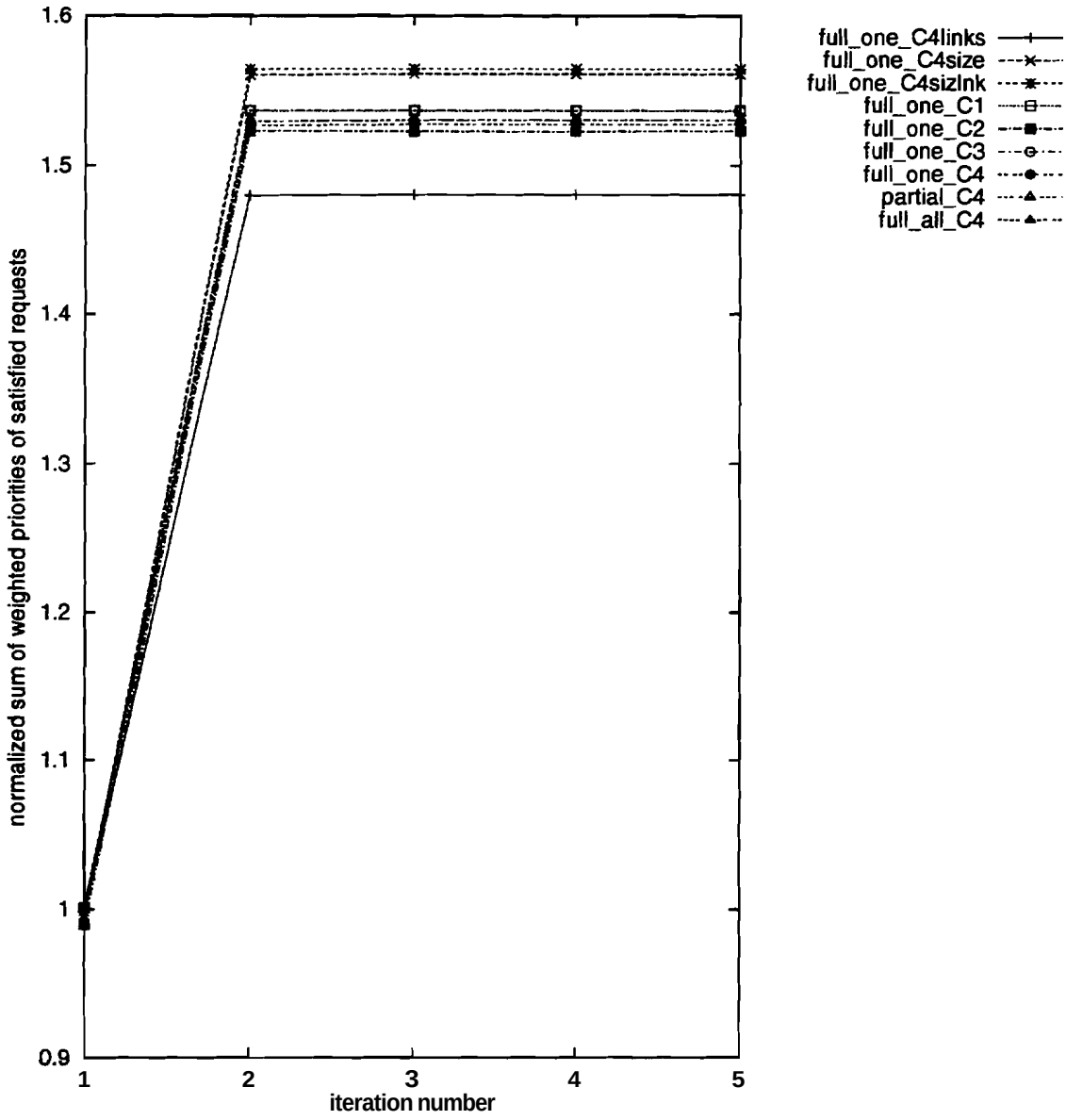

Figure 9.33: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.2, an average link traversal count of 2.5, and an ω value of 16.

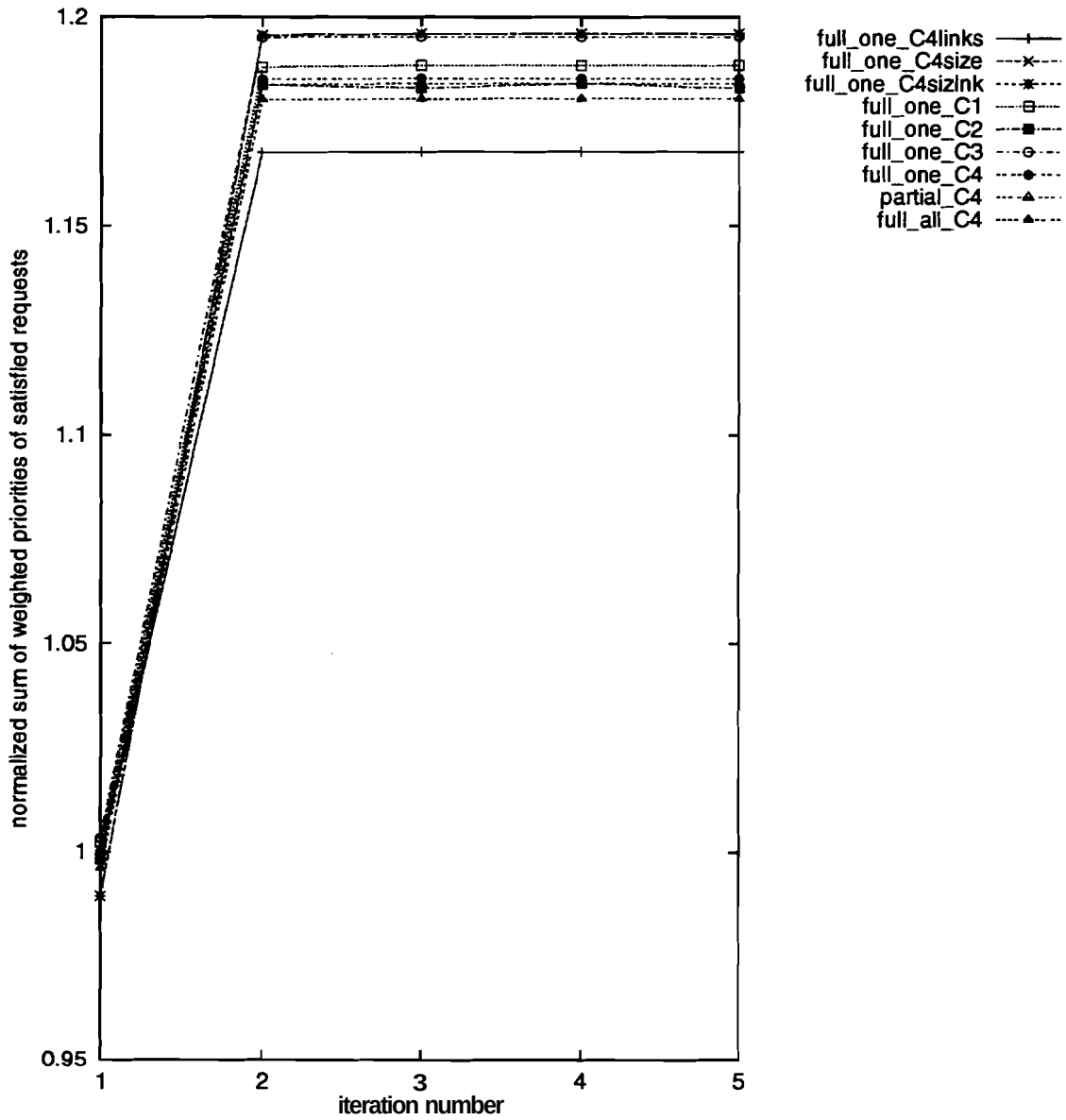

Figure 9.34: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.4, an average link traversal count of 2.5, and an w value of 16.

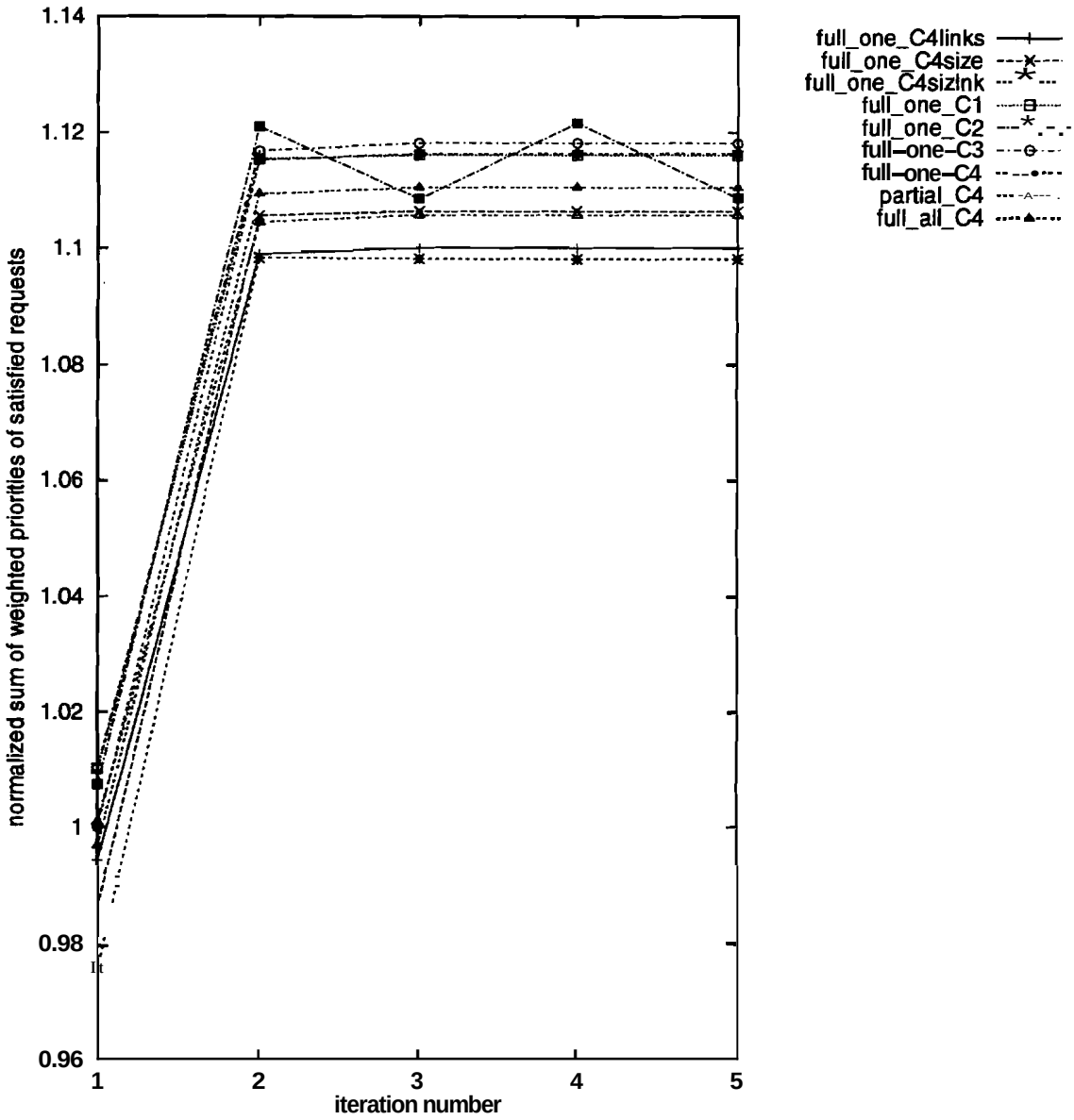

Figure **9.35:** Weighted sum of satisfied requests' priorities normalized to the performance of full–one–C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 0.8, an average link traversal count of **2.5,** and an *w* value of 16.

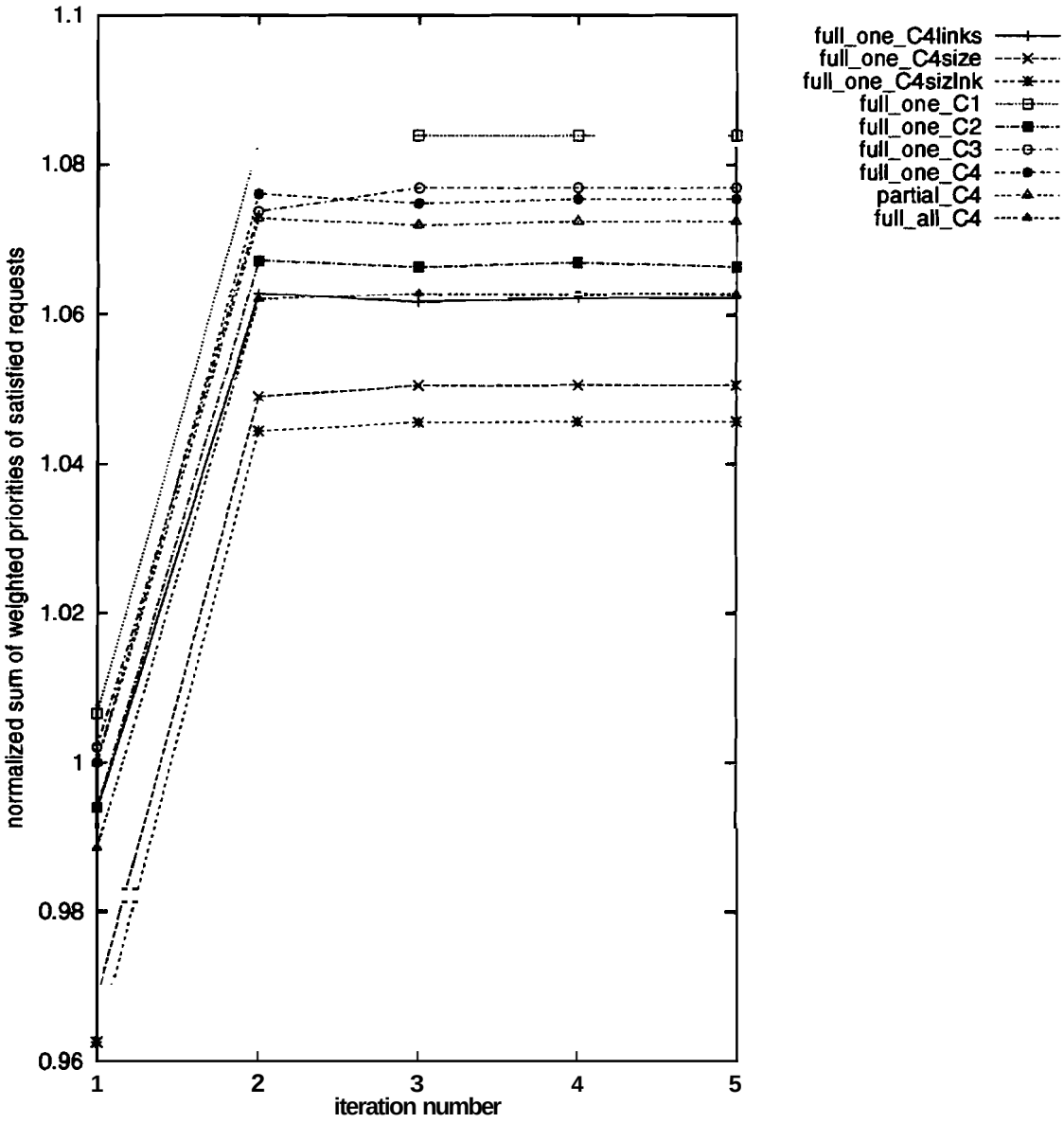

Figure 9.36: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 1.6, an average link traversal count of 2.5, and an w value of 16.

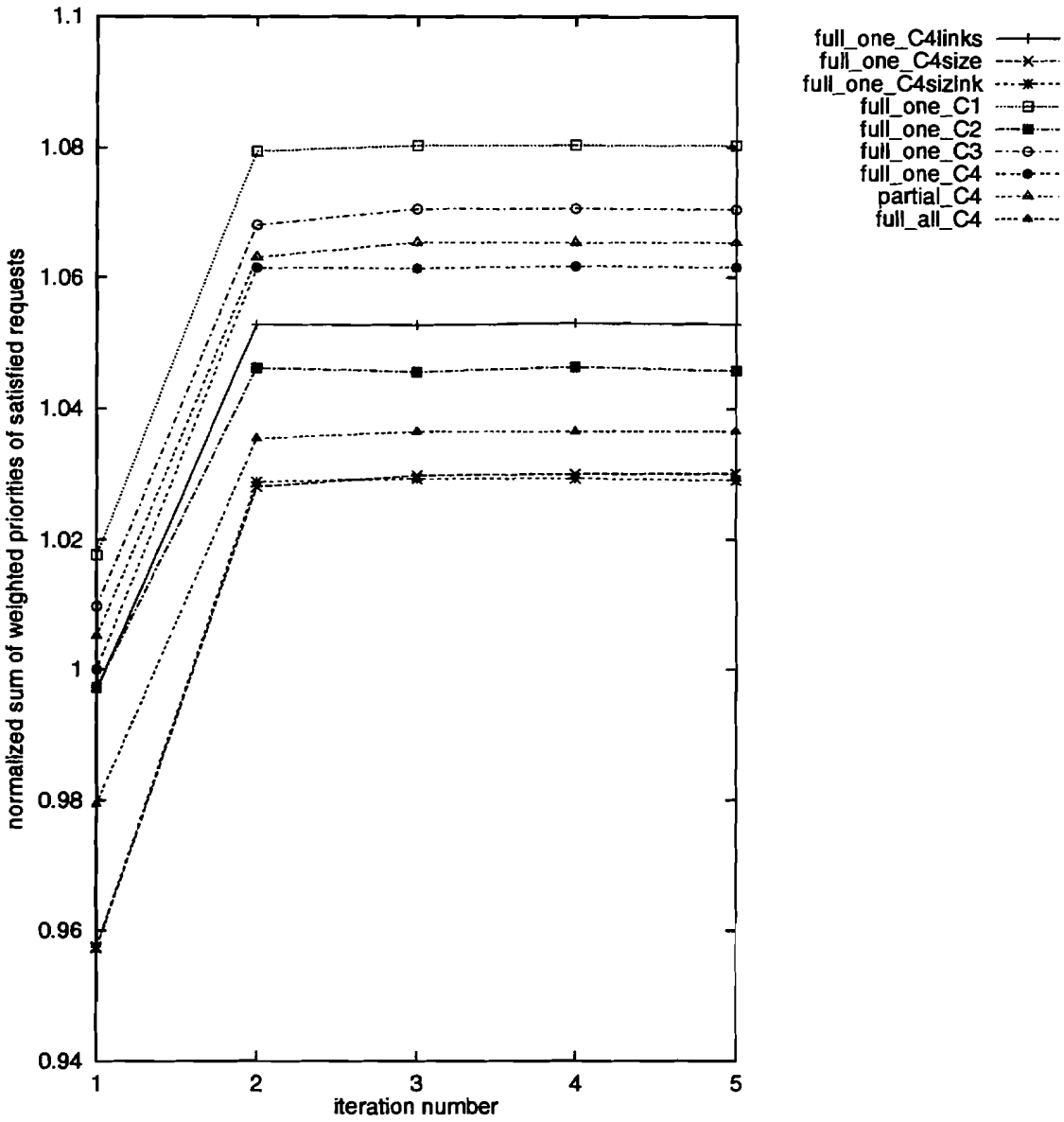

Figure 9.37: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 3.1, an average link traversal count of 2.5, and an *w* value of 16.

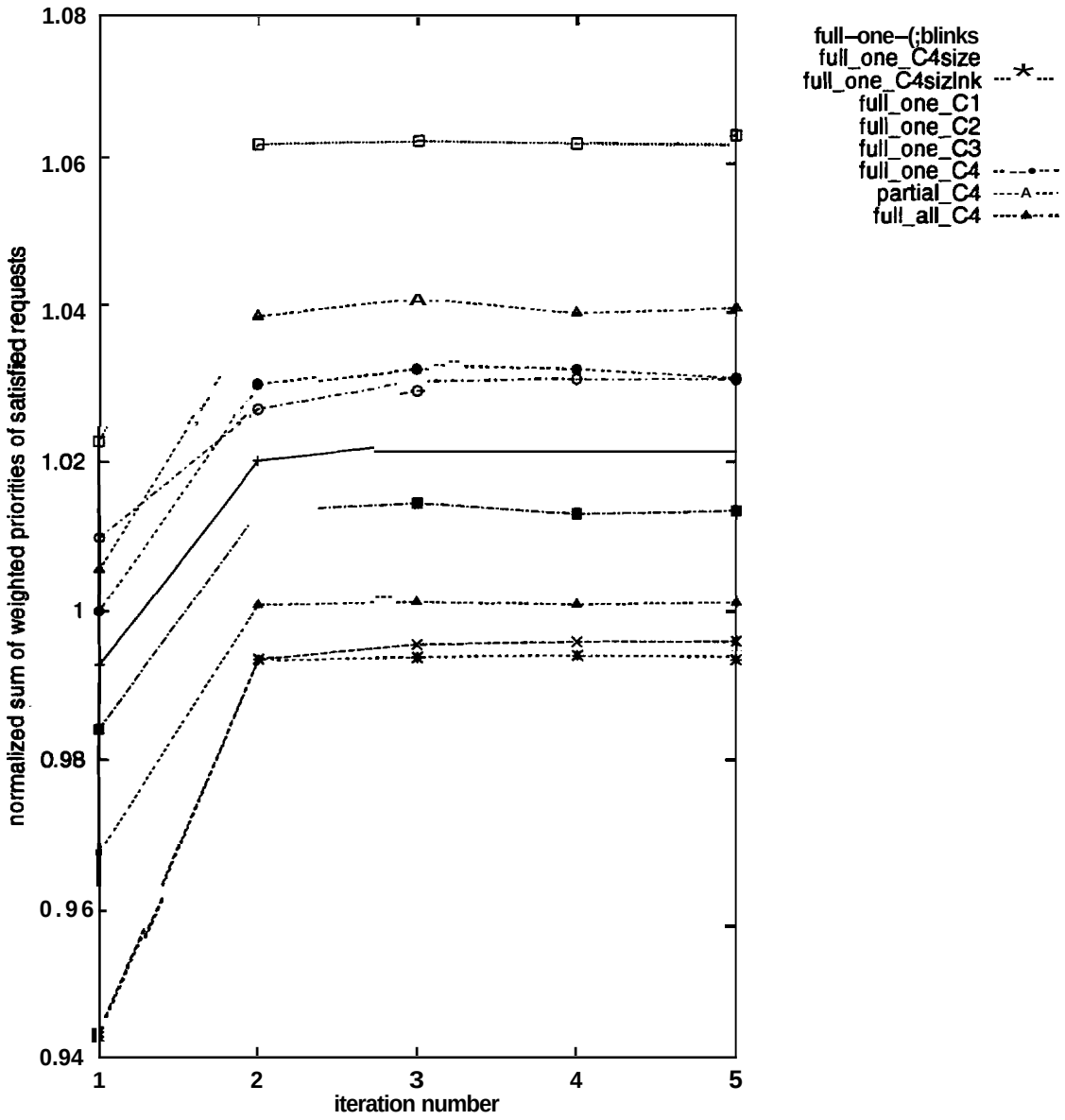

Figure 9.38: Weighted sum of satisfied requests' priorities normalized to the performance of full–one–C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 6.2, an average link traversal count of 2.5, and an *w* value of 16.

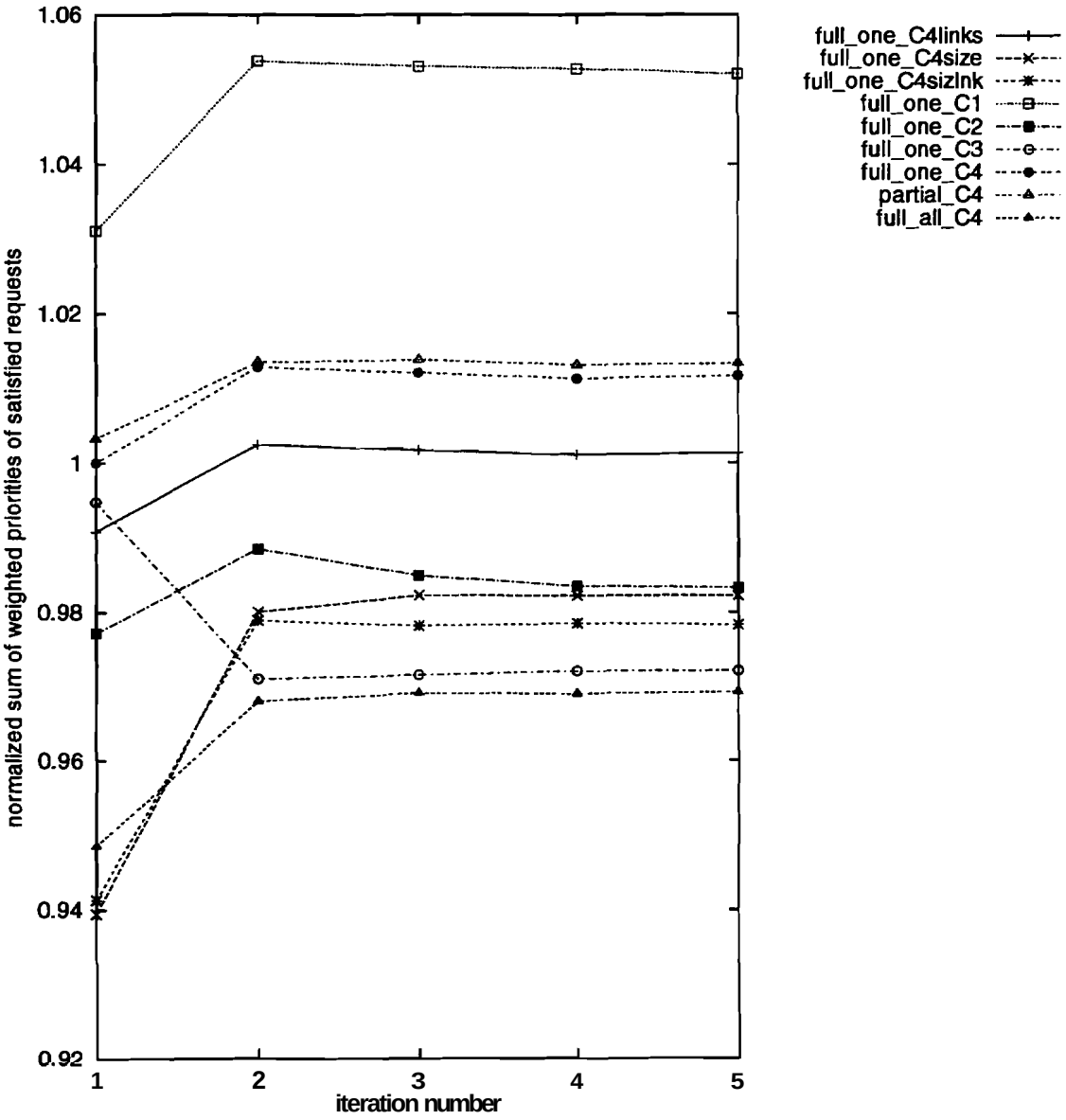

Figure 9.39: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 12.5, an average link traversal count of 2.5, and an ω value of 16.

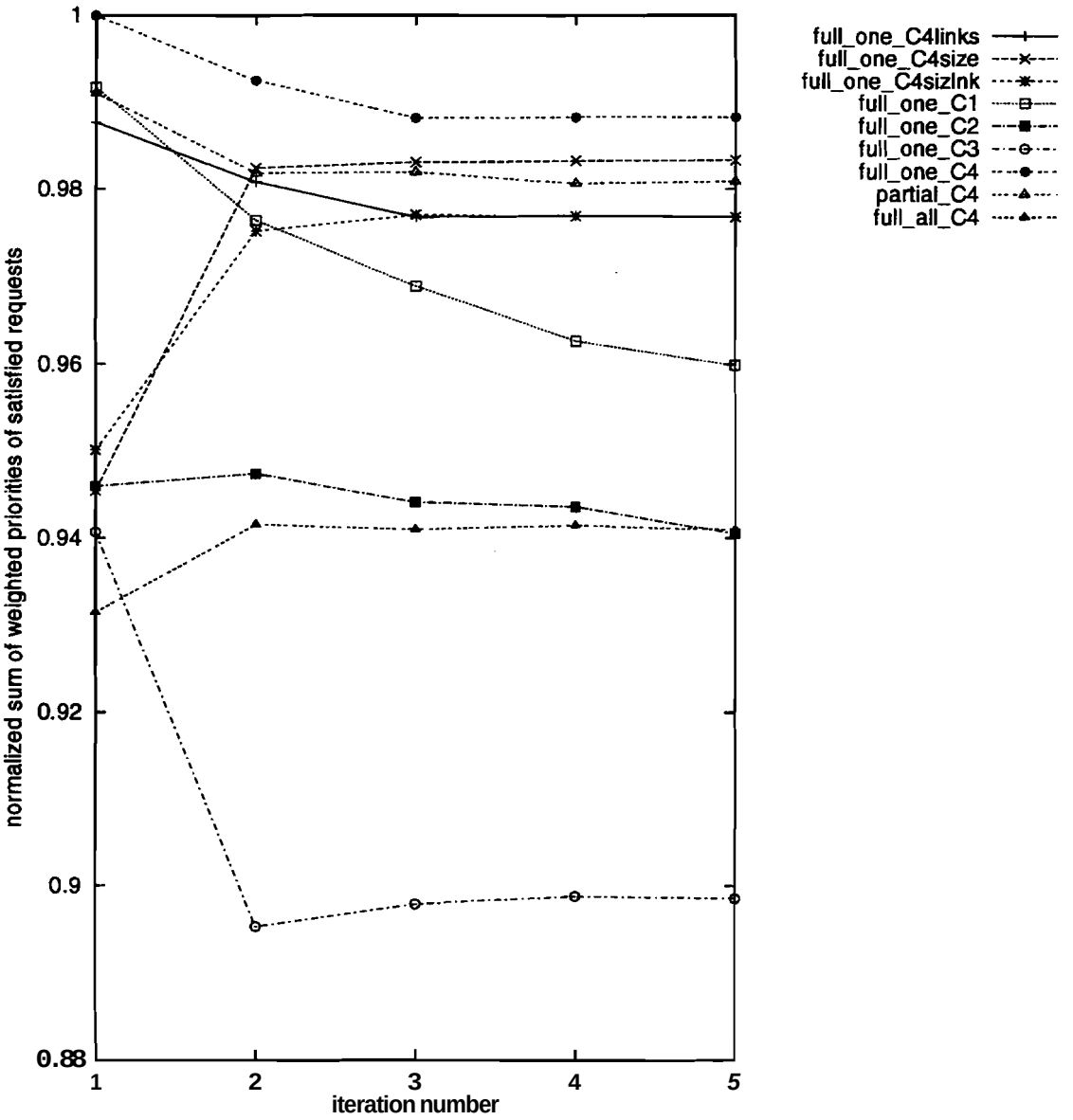

Figure 9.40: Weighted sum of satisfied requests' priorities normalized to the performance of full−one−C4 in iteration 1. Shown here is the performance of each heuristic after each iteration of the variable time, variable accuracy algorithm. The data set had an oversubscription rate of 25.0, an average link traversal count of 2.5, and an ω value of 16.

10. Summary and Conclusions

Data staging is an important data management issue for distributed computer systems. It addresses the issues of distributing and storing over numerous geographically dispersed locations both repository data and continually generated data through an oversubscribed network, where not all data requests can be satisfied. When certain data with their corresponding priorities need to be collected together at a site with limited storage capacities in a timely fashion, a heuristic must be devised to schedule the necessary communication steps efficiently.

The performance of fourteen heuristics were shown, and compared to three upper bounds and three lower bounds. Many different weighting schemes for the relative importance of different priority levels of requested data items were considered. Each procedure and cost criterion was designed with particular advantages in mind. The results presented showed that, for the system parameters considered (e.g., priority weighting, oversubscription rate), the combination of cost $C4$ or $C1$ with the full path/one destination procedure consistently performed the best, when using the measure of weighted sum of priorities satisfied.

Because each heuristic has advantages, the pair that performs best may differ depending on the system parameters (i.e., the actual environment where the scheduler heuristic will be deployed). Extensive work was done exploring the performance of the heuristics when varying the congestion of the network, the number of virtual links used to satisfy data items, and the priority weighting scheme. In summary, a class of heuristics and cost criteria that compare well to upper and lower bounds were developed and analyzed. While in general several heuristics perform comparably, if a system is known to have a particular operating environment (e.g., w value, oversubscription rate), there may be a preference for one pair over another.

An additional novel approach using a variable time, variable accuracy method that

considered multiple data item versions with different resource requirements was evaluated. The use of multiple versions was shown to help some heuristics in all but the most oversubscribed cases; in many cases, the improvement was over 10%.

Studies such as the one presented in this report are necessary to design schedules for distributed communication systems. Numerous heuristics were shown to perform very effectively. The exact heuristic to use in a given situation will depend on the system operator environment.

Acknowledgments: The authors thank Joe Rockmore, Bob Beaton, Jose Fortes, and Edwin Chong for their valuable comments and suggestions.

GLOSSARY

$A_T[i,j]$ the earliest possible time found so far when $Rq[i]$ is available on $M[j]$ (where $0 \leq i < 2\rho$ and $0 \leq j < m$)

$C1[i,j][s,r][k]$ the cost for sending data item $Rq[i]$ to $M[r]$ from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine:

$$-W_E * Efp[i,j] - W_U * Urgency[i,j]$$

(where $0 \leq i < 2\rho$ and $0 \leq r < m$ and $(s,k) = \pi[i,r]$ and $0 \leq j < Nrq[i]$)

$C2[i][s,r][k]$ the cost for sending data item $Rq[i]$ to $M[r]$ from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine(s):

$$-W_E * \left(\sum_{j \in Drq[i,r]} Efp[i,j] \right) - W_U * \left(\max_{j \in Drq[i,r]} Urgency[i,j] \right)$$

(where $0 \leq i < 2\rho$ and $0 \leq r < m$ and $(s,k) = \pi[i,r]$)

$C3[i][s,r][k]$ the cost for sending data item $Rq[i]$ to $M[r]$ from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine(s):

$$\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{Urgency[i,j]}$$

(where $0 \leq i < 2\rho$ and $0 \leq r < m$ and $(s,k) = \pi[i,r]$)

$C4[i][s,r][k]$ the cost for sending data item $Rq[i]$ to $M[r]$ from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine(s):

$$-W_E * \left(\sum_{j \in Drq[i,r]} Efp[i,j] \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right)$$

(where $0 \leq i < 2\rho$ and $0 \leq r < m$ and $(s,k) = \pi[i,r]$)

$C4links[i][s,r][k]$ the cost for sending data item $Rq[i]$ to $M[r]$ from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine(s):

$$-W_E * \left(\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{Nlinks[i,j]} \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right)$$

(where $0 \leq i < 2\rho$ and $0 \leq r < m$ and $(s,k) = \pi[i,r]$)

$C4size[i][s,r]$[k] the cost for sending data item $Rq[i]$ to M[r] from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine(s):

$$-W_E * \left(\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{|Rq[i]|} \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right)$$

(where $0 \leq$ i $<$ 2ρ and $0 \leq$ r $<$ m and (s, k) $= \pi[i,r]$)

$C4sizlnk[i][s,r][k]$ the cost for sending data item $Rq[i]$ to $M[r]$ from $M[s]$ via link $L[s,r][k]$, in order to ultimately try to satisfy the jth requesting destination machine(s):

$$-W_E * \left(\sum_{j \in Drq[i,r]} \frac{Efp[i,j]}{|Rq[i]| * Nlinks[i,j]} \right) - W_U * \left(\sum_{j \in Drq[i,r]} Urgency[i,j] \right)$$

(where $0 \leq i < 2\rho$ and $0 \leq$ r $<$ m and $(s,k) = \pi[i,r]$)

$Cap[i](t_j)$ constant unused storage capacity of machine $M[i]$ during the time interval $[t_j, t_{j+1})$ (where $0 \leq i <$ m)

set of n data items with unique names that are available on the machines in M

$\delta[i]$ an individual data item in **A** (where $0 \leq i <$ n)

$|\delta[i]|$ the size of data item $\delta[i]$ (where $0 \leq i <$ n)

Srt$[l,$ i] the removal time that data item $\delta[l]$ can be removed from machine $M[i]$ (where $0 \leq l <$ n and $0 \leq$ i $<$ m)

$\delta st[l,j]$ the start time that data item $\delta[l]$ becomes available at its jth source machine (where $0 \leq l < n$ and $0 \leq$ j $< N\delta[l]$)

$D[i,j][k]$ $(|\delta[l]|)$ time duration required to transfer data item $\delta[l]$ from machine $M[i]$ to $M[j]$ via the virtual link $L[i,j][k]$ (where i $\neq$ j and $0 \leq i,$ j $<$ m and $0 \leq k < Nl[i,j]$ and $0 \leq l < n$)

$Drq[i,r]$ set of indices of machines that request $Rq[i]$ along a path through $M[r]$, where $M[r]$ is the next machine to receive $Rq[i]$ (where $0 \leq i < 2\rho$ and $0 \leq r < m$)

$E[S_h]$ the effect of the schedule S_h; minimizing this over all values of h is the global optimization criterion:

$$-\left(\sum_{(j,k) \in Srq[S_h]} W[Priority[j,\text{k}]] * Worth[j,\text{k}] \right)$$

(where $0 \leq$ h $< a$ and $0 \leq j < 2\rho$ and $0 \leq k < Nrq[j]$)

$Efp[i,k]$	effective priority of data item $Rq[i]$ at the kth requesting location: $Sat[i,k] * W[Priority[i,k]] * Worth[i,k]$ (where $0 \leq i < 2\rho$ and $0 \leq k < Nrq[i]$)
	time period for intermediate storage machines to hold a data item after the last deadline for that data item has expired
G_{nt}	network topology graph composed of a set of vertices representing machines M and edges representing links L
L	the set of virtual links in G_{nt}
$L[i,j][k]$	the kth virtual link from machine $M[i]$ to $M[j]$ (where $i \neq$ j and $0 \leq i,j < m$ and $0 \leq k < Nl[i,j]$)
$Lst[i,j][k]$	link start time that the kth virtual link from machine $M[i]$ to $M[j]$ becomes available (where i $\neq$ j and $0 \leq i,j < m$ and $0 \leq k < Nl[i,j]$)
$Let[i,j][k]$	link end time that the kth virtual link from machine $M[i]$ to $M[j]$ becomes unavailable (where $i \# j$ and $0 \leq i,j < m$ and $0 \leq k < Nl[i,j]$)
m	number of machines in the set M
M	the set of machines in G_{nt}
$M[i]$	an individual machine in M (where $0 \leq i < m$)
n	number of distinctive data items in **A**
$N\delta[l]$	number of source machines holding a copy of $\delta[l]$ (where $0 \leq l < n$)
$NetBandwidth$	sum of the total number of bytes that could be transmitted over each virtual link in the system
$Nl[i,j]$	number of virtual links from machine $M[i]$ to $M[j]$ (where $i \neq j$ and $0 \leq i,j < m$)
$Nlinks[i,k]$	number of virtual links used to get from any $M[s]$, which holds a copy of $Rq[i]$, to destination $M[Request[i,k]]$ using the most recent path generated by Dijkstra's algorithm (where $0 \leq s < m$ and $0 \leq i < 2\rho$ and $0 \leq k < Nrq[i]$)
$Nrq[j]$	number of destination machines that request $Rq[j]$ (where $0 \leq j < 2\rho$)
$\pi[i,j]$	the two-tuple (s,k) identifying the machine $M[s]$ that sends $Rq[i]$ to $M[j]$ via virtual link $L[s,j][k]$ (where $0 \leq i < 2\rho$ and $0 \leq j < m$ and $-1 \leq s <$ m and $-1 \leq k < Nl[s,j]$)
P	the most important priority class
$Priority[j,k]$	priority class of $Rq[j]$ at requesting destination $M[Request[j,k]]$ (where $0 \leq$ j $< 2\rho$ and $0 \leq k < Nrq[j]$ and $0 \leq Priority[j,k] \leq P$)

	number of unique higher quality data items in Rq, also the number of unique lower quality data items in Rq, for a total of 2ρ data items in Rq
$ReqBandwidth$	sum over all data requests of the number of bytes of bandwidth needed to satisfy each request when considered individually
$Request[j,k]$	index of the kth machine that requested $Rq[j]$ (where $0 \leq j < 2\rho$ and $0 \leq k < Nrq[j]$ and $0 \leq Request[j,k] < m$)
$Rft[j,k]$	deadline time after which data item $Rq[j]$ is no longer useful to machine $M[Request[j,k]]$ (where $0 \leq j < 2\rho$ and $0 \leq k < Nrq[j]$)
Rq	the set of requested data items; two versions of each data item are present
$Rq[j]$	a requested data item (where $0 \leq j < 2\rho$); higher quality data items have j in the range $0 \leq j < \rho$, lower quality data items have j in the range $\rho \leq j < 2\rho$
S	set of schedules for the communication steps within the network
S_h	a schedule consisting of a series of communication steps among the machines of M using the communication links in L (where $0 \leq$ h $< \sigma$)
$Sat[ik]$	1 if $Request[i,k]$ would be satisfiable using current network information, and 0 if it would not be satisfiable (where $0 \leq i < 2\rho$ and $0 \leq k < Nrq[i]$)
$Source[i,j]$	index of the jth source machine for data item $\delta[i]$ (where $0 \leq i < n$ and $0 \leq j < N\delta[i]$ and $0 \leq Source[i,j] < m$)
$Srq[S_h]$	set of two-tuples (j,k) such that the kth request for the data item $Rq[j]$ is satisfiable with respect to the schedule S_h (where $0 \leq j < 2\rho$ and $0 \leq k < Nrq[j]$ and $0 \leq$ h $< a$)
$Urgency[i,k]$	measure of closeness of a data item's available time to its deadline time (in seconds) at a requesting destination: $$-Sat[i,k] * (Rft[i,k] - A_T[i, Request[i,k]] + 1)$$ (where $0 \leq i < 2\rho$ and $0 \leq k < Nrq[i]$)
$W[\alpha]$	the relative weight of a priority class α (where $0 \leq \alpha \leq P$)
W_E	the weight of the effective priority term in the scheduling cost functions
W_U	the weight of the urgency term in the scheduling cost functions
$Worth[j,k]$	a percentage of value to a user of data item $Rq[j]$ sent to satisfy a request at $M[Request[j,k]]$; assumes value 1 if $j < p$ and $Rq[j]$ is used to satisfy $M[Request[j,k]]$, assumes value 0.25 if $j \geq \rho$ and $Rq[j]$ is used to satisfy $M[Request[j,k]]$, and assumes value 0 otherwise (where $0 \leq j < 2\rho$ and $0 \leq k < Nrq[j]$)

LIST OF REFERENCES

[AcZ93] S. Acharya and S. B. Zdonik, "An efficient scheme for dynamic data replication," Technical Report CS-93-43, Department of Computer Science, Brown University, September 1993, 25 pp.

[BaB97] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, "Enhancing the web's infrastructure: From caching to replication," IEEE Internet Computing, Vol. 1, No. 2, March-April 1997, pp. 18-27.

[BaO98] S. Balakrishnan and F. Özgüner, "A priority-driven flow control mechanism for real-time traffic in multiprocessor networks," IEEE Transactions on Parallel and Distributed Systems , Vol. 9, No. 2, July 1998, pp. 664678.

[Bes97] A. Bestavros, "WWW traffic reduction and load balancing through server-based caching," IEEE Concurrency, Vol. 5, No. 1, January-March 1997, pp. 56-67.

[Bec99] N. B. Beck, Design and Evaluation of Heuristics for Data Staging in a Distributed Communication Network, Master of Science Thesis, School of Electrical and Computer Engineering, Purdue University, May 1999.

[BrS98] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," IEEE Workshop on Advances in Parallel and Distributed Systems, October 1998, pp. 330-335 (included in the proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, October 1998).

[BrS99] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," 8th IEEE Workshop on Heterogeneous Computing Systems *(HCW '99)*, April 1999, pp. 15-29.

[Cas93] C. G. Cassandras, Discrete Event Systems: Modeling and Performance Analysis, Irwin, Homewood, IL, 1993.

[ChD81] R. Chandrasekaran and A. Dauchety, "Location on tree networks: P-centre and n-dispersion problems," Mathematics of Operations Research, Vol. 6, No. 1, February 1981, pp. 50-57.

[CoL90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[CoN80] G. Cornuejols, G. L. Nemhauser, and L. A. Wolsey, 'Worst-case and probabilistic analysis of algorithms for a location problem," Operations Research, Vol. 28, No. 4, July-August 1980, pp. 847-858.

[DaH93] P. Danzig, R. Hall, and M. Schwartz, "A case for caching file objects inside internetworks," Technical Report CU-CS-642-93, Computer Science Department, University of Colorado, March 1993, 15 pp.

[HeK99] D. Hensgen, T. Kidd, D. St. John, M. Schnaidt, H. J. Siegel, T. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: The management system for heterogeneous networks," 8th *IEEE* Workshop on Heterogeneous Computing Systems *(HCW'99)*, April 1999, pp. 184-198.

[HuM89] A. P. Hurter and J. S. Martinich, Facility Location and The Theory of Production, Kluwer Academic Publishers, Norwell, MA, 1989.

[JoL95] P. C. Jones, T. J. Lowe, G. Muller, N. Xu, Y. Ye, and J. L. Zydiak, "Specially structured uncapacitated facility location problem," Operations Research, Vol. 43, No. 4, July-August 1995, pp. 661-669.

[LeB97] M. J. Lemanski and J. C. Benton, Simulation for *SmartNet* Scheduling of Asynchronous Transfer Mode Virtual Channels, Master of Science Thesis, Department of Computer Science, Naval Postgraduate School, June 1997 (Advisor: D. Hensgen).

[MaA99] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," 8th *IEEE* Workshop on Heterogeneous Computing Systems (HCW *'99)*, April 1999, pp. 30-44.

[MoC84] I. D. Moon and S. S. Chaudhry, "An analysis of network location problems with distance constraints," Management Science, Vol. 30, No. 3, March 1984, pp. 290-307.

[Roc96] A. J. Rockmore, "BADD functional description," Internal DARPA Memo, February 1996, 9 pp.

[Shi77] D. R. Shier, "A min-max theorem for pcenter problems on a tree," Transportation Science, Vol. 11, No. 3, August 1977, pp. 243-252.

[Sma96] SmartNet/Heterogeneous Computing Team, "BC2A/TACITUS/BADD integration plan," Internal NRaD Naval Laboratory Report, August 1996, 16 pp.

[TaS97] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE* Transactions on Parallel and Distributed Systems, Vol. 8, No. 8, August 1997, pp. 857-871.

[TaS98] M. Tan and H. J. Siegel, "A stochastic model for heterogeneous computing and its application in data relocation scheme development," *IEEE* Transactions on Parallel and Distributed Systems, Vol. 9, No. 11, November 1998, pp. 1088-1101.

[TaT98] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, M. Jurczyk, "A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment," Proceedings of the 7th *IEEE* Workshop on Heterogeneous Computing Systems *(H*CW *'98)*, April 1998, pp. 115-129.

[ThT99] M. D. Theys, M. Tan, N. B. Beck, H. J. Siegel, M. Jurczyk, "Heuristics and a Mathematical Framework for Scheduling Data Requests in a Distributed Communication Network," Technical Report TR-ECE 99-2, School of Electrical and Computer Engineering, Purdue University, January 1999, 58 pp.

[WaS97] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," Journal of Parallel and Distributed Computing, Vol. 47, No. 1, Nov. 1997, pp. 1-15.