Bicriteria data compression*

Andrea Farruggia, Paolo Ferragina, Antonio Frangioni, and Rossano Venturini

Dipartimento di Informatica, Università di Pisa, Italy

{farruggi, ferragina, frangio, rossano}@di.unipi.it

Abstract

In this paper we address the problem of trading optimally, and in a principled way, the compressed size/decompression time of LZ77 parsings by introducing what we call the *Bicriteria LZ77-Parsing problem.*

The goal is to determine an LZ77 parsing which minimizes the space occupancy in bits of the compressed file, provided that the decompression time is bounded by T. Symmetrically, we can exchange the role of the two resources and thus ask for minimizing the decompression time provided that the compressed space is bounded by a fixed amount given in advance.

We address this goal in three stages: (i) we introduce the novel Bicriteria LZ77-Parsing problem which formalizes in a principled way what data-compressors have traditionally approached by means of heuristics; (ii) we solve this problem efficiently, up to a negligible *additive* constant, in $O(n \log^2 n)$ time and optimal $O(n)$ words of working space, by proving and deploying some specific structural properties of a weighted graph derived from the possible LZ77-parsings of the input file; (iii) we execute a preliminary set of experiments which show that our novel compressor is very competitive to all the highly engineered competitors (such as Snappy, lzma, bzip2), hence offering a win-win situation in theory&practice.

1 Introduction

The advent of massive datasets and the consequent design of high-performing distributed storage systems—such as BigTable by Google [7], Cassandra by Facebook [5], Hadoop by Apache—have reignited the interest of the scientific and engineering community towards the design of lossless data compressors which achieve effective compression ratio and very efficient decompression speed. The literature abounds of solutions for this problem, named "compress once, decompress many times", that can be cast into two main families: the compressors based on the Burrows-Wheeler Transform [6], and the ones based on the Lempel-Ziv parsing scheme [35, 36]. Compressors are known in both families that require time linear in the input size, both for compressing and decompressing the data, and take compressed-space which can be bound in terms of the k-th order empirical entropy of the input [25, 35].

But the compressors running behind those large-scale storage systems are not derived from those scientific results. The reason relies in the fact that theoretically efficient compressors are optimal in the RAM model, but they elicit many cache/IO misses during the decompression step. This poor behavior is most prominent in the BWT-based compressors, and it is not negligible in the LZ-based approaches. This motivated the software engineers to devise variants of Lempel-Ziv's original proposal (e.g. Snappy, LZ4) with the injection of several software tricks which have beneficial effects on memory-access locality. These compressors expanded further the known jungle of space/time trade-offs[1], thus posing the software engineers in front of a choice: either achieve effective/optimal compression-ratios, possibly sacrificing the decompression speed (as it occurs in the theory-based results [15–17]); or try to balance them by adopting a plethora of programming tricks which trade compressed space by decompression time (such as in Snappy, LZ4 or in the recent LZ77-end [26]), thus waiving mathematical guarantees on their final performance.

In the light of this dichotomy, it would be natural to ask for an algorithm which guarantees effective compression-ratios and efficient decompression speed in hierarchical memories. In this paper, however, we aim for a more ambitious goal which is further motivated by the following two simple, yet challenging, questions:

- who cares whether the compressed file is slightly

*This work was partially supported by MIUR of Italy under projects PRIN ARS Technomedia 2012 and the eCloud EU Project.

[1] See e.g., `http://mattmahoney.net/dc/`.

longer than the one achievable with BWT-based compressors, provided that we can improve significantly BWT's decompression time? This is a natural question arising in the context of distributed storage-systems, and the one leading the design Snappy and LZ4.

- who cares whether the compressed file can be decompressed slightly slower than Snappy or LZ4, provided that we can improve significantly their compressed space? This is a natural question in a context where space occupancy is a major concern, e.g., tablets and smart-phones, and the one for which tools like Google's Zopfli have been recently introduced.

If we are able to offer mathematical guarantees to the meaning of "slightly longer/slower", then these two questions become pertinent and challenging in theory too. So in this paper we introduce the following problem, that we call *bicriteria data compression*: given an input file $\mathcal{S}$ and an upper bound T on its decompression time, the goal is to determine a compressed version of $\mathcal{S}$ which minimizes the compressed space provided that it can be decompressed in T time. Symmetrically, we could exchange the role of time/space resources, and thus ask for the compressed version of $\mathcal{S}$ which minimizes the decompression time provided that the compressed-space occupancy is within a fixed bound.

In order to attack this problem in a principled way we need to fix two ingredients: the class of compressed versions of $\mathcal{S}$ over which this bicriteria optimization will take place; and the computational model measuring the resources to be optimized. For the former ingredient we will take the class of *LZ77-based compressors* because they are dominant in the theoretical (e.g., [8, 9, 14, 17, 22]) and in the practical setting (e.g., gzip, 7zip , Snappy, LZ4, [24, 26, 34]). In Section 2, we will show that the Bicriteria data-compression problem formulated over LZ77-based compressors is well funded because there exists an *infinite class of strings* which can be parsed in many different ways, thus offering a wide spectrum of space-time trade-offs in which small variations in the usage of one resource (e.g., time) may induce arbitrary large variations in the usage of the other resource (e.g., space).

For the latter ingredient, we take inspiration from several models of computation which abstract multi-level memory hierarchies and the fetching of contiguous memory words [1, 2, 4, 28, 33]. In these models the cost of fetching a word at address x takes $f(x)$ time, where $f(x)$ is a non-decreasing, polynomially bounded function (e.g., $f(x) = \lceil \log x \rceil$ and $f(x) = x^{O(1)}$). Some of these models offer also a *block copy* operation, in which a sequence of ℓ consecutive words can be copied from memory location x to memory location y (with $x \geq y$) in time $f(x) + \ell$. We remark that, in our scenario, this model is more proper than the frequently adopted two-level memory model [3], because we care to differentiate between contiguous/random accesses to memory-disk blocks, which is a feature heavily exploited in the design of modern compressors [10].

Given these two ingredients, we devise a formal framework that allows us to analyze any LZ77-parsing scheme in terms of both the space occupancy (in bits) of the compressed file, and the time cost of its decompression taking into account the underlying memory hierarchy (see Section 3). More in detail, we will extend the model proposed in [17], based on a special weighted DAG consisting of $n = |\mathcal{S}|$ nodes, one per character of $\mathcal{S}$, and $m = O(n^2)$ edges, one per possible phrase in the LZ77-parsing of $\mathcal{S}$. In our new graph each edge will have attached two weights: a *time weight*, that accounts for the time to decompress a phrase (derived according to the hierarchical-memory model mentioned above), and a *space cost*, that accounts for the number of bits needed to store the LZ77-phrase associated to that edge (derived according to the integer-encoder adopted in the compressor). Every path π from node 1 to node n in $\mathcal{G}$ (hereafter, named "1n-path") corresponds to an LZ77-parsing of the input file $\mathcal{S}$ whose *compressed-space occupancy* is given by the sum of the *space-costs* of π's edges (say $s(\pi)$) and whose *decompression-time* is given by the sum of the *time-weights* of π's edges (say $t(\pi)$). As a result of this correspondence, we will be able to rephrase our *bicriteria LZ77-parsing* problem into the well-known *weight-constrained shortest path problem* (WCSPP) (see [30] and references therein) over the weighted DAG $\mathcal{G}$, in which the goal will be to search for the 1n-path π whose decompression-time is $t(\pi) \leq T$ and whose compressed-space occupancy $s(\pi)$ is minimized. Due to its vast range of applications, WCSPP received a great deal of attention from the optimization community. It is an $\mathcal{NP}$-Hard problem, even on a DAG with positive weights and costs [11, 18], and it can be solved in pseudo-polynomial $O(mT)$ time via dynamic programming [27]. Our version of the WCSPP problem has m and T bounded by $O(n \log n)$ (see Section 3), so it can be solved in polynomial time, namely $O(mT) = O(n^2 \log^2 n)$ time and $O(n^2 \log n)$ space. Unfortunately this bounds are unacceptable in practice, because $n^2 \approx 2^{64}$ just for one Gb of data to be compressed.

The second contribution of this paper is to prove some structural properties of our weighted DAG which allow us to design an algorithm that approximately solves our version of WCSPP in $O(n \log^2 n)$ time and $O(n)$ working space. The approximation is *additive*, that is, our algorithm determines a LZ77-parsing whose decompression time is $\leq T + 2\,t_{\max}$ and whose compressed space is just $s_{\max}$ bits more than the optimal one, where $t_{\max}$ and $s_{\max}$ are, respectively, the maximum time-weight and the maximum space-cost of any edge in the DAG. Given that the values of $s_{\max}$ and $t_{\max}$ are logarithmic in n (see Section 2), those additive terms are negligible. We remark here that this type of additive-approximation is clearly related to the *bicriteria-approximation* introduced by [29], and it is more desirable than the "classic" (α, β)-approximation [20] because ours is additive whereas the latter is multiplicative, so the larger is the problem size the better is our approximation. The further peculiarity of our approach is that we are using the additive-approximation to speed-up the solution to a problem that already admits in our setting a polynomial solution which, however, grows as $\Omega(n^2)$ thus resulting unusable in practice.

The third, and last, contribution of this paper is to present a set of preliminary experimental results which compare an implementation of our compressor against state-of-the-art LZ77-based algorithms (Snappy, LZMA, LZ4, gzip) and BWT-based algorithms (with bounded and unbounded memory footprint). These experiments bring out two key aspects: (i) they provide a practical ground to the two pertinent questions posed at the beginning of the paper, thus, motivating the theoretical analysis introduced with our novel Bicriteria data-compression problem; (ii) they show that our parsing strategy dominates all the highly engineered competitors, by exhibiting decompression speeds close to those of Snappy and LZ4 (i.e., the fastest known ones), and compression ratios close to those of BWT-based and LZMA compressors (i.e., the more succinct ones).

2 On the LZ77-parsing

Let $\mathcal{S}$ be a string of length n built over an alphabet $\Sigma = [\sigma]$ and terminated by a special character. We denote by $\mathcal{S}[i]$ the i-th character of $\mathcal{S}$ and by $\mathcal{S}[i, j]$ the substring ranging from i to j (included). The compression algorithm LZ77 works by *parsing* the input string $\mathcal{S}$ into phrases $p_1, \ldots, p_k$ such that phrase p_i can be any substring of $\mathcal{S}$ starting in the prefix $p_1, \ldots, p_{i-1}$. Once the parsing has been identified, each phrase is represented via *codewords*, that are pairs of integers $\langle d, \ell\rangle$, where d is the distance from the position where the copied phrase occurs, and ℓ is its length. Every first occurrence of a new character c is encoded as $\langle 0, c\rangle$. These pairs are compressed via variable-length integer encoders which eventually produces the compressed output of $\mathcal{S}$ as a sequence of bits. Among all possible parsing strategies, the *greedy parsing* is widely adopted: it chooses p_i as the longest prefix of the remaining suffix of $\mathcal{S}$. This is optimal whenever the goal is to minimize the number of generated phrases or, equivalently, the phrases have equal bit-length; but if phrases are encoded with a variable number of bits then the greedy approach may be sub-optimal [17].

Modeling the space occupancy. A LZ77-phrase $\langle d, \ell\rangle$ is typically compressed by using two distinct (universal) integer encoders, since distances d and lengths ℓ are distributed differently in $\mathcal{S}$. We use $s(d, \ell)$ to denote the length in bits of the encoding of $\langle d, \ell\rangle$. We restrict our attention on variable-length integer encoders which emit longer codewords for bigger integers, the so called *non-decreasing cost property.* This assumption is not restrictive because it encompasses all universal encoders, such as Truncated binary, Elias' Gamma and Delta [12], Golomb [19], and LZ4's encoder. An interesting fact about these encoders is that they take a logarithmic number of bits per integer. This fact is crucial in evaluating the complexity of our algorithm, since it depends on the number of distinct values assumed by $s(d, \ell)$ when $d, \ell \leq n$. We denote by s_{costs} this number, which is $O(\log n)$ for all the universal encoders above.

For the sake of presentation, we denote by $s(\pi)$ the bit-length of the compressed output generated according to the LZ77-parsing π. This is estimated by summing the lengths of the encoding of all phrases in π, hence $\sum_{\langle d,\ell\rangle \in \pi} s(d, \ell)$.

Modeling the decompression speed. The aim of this section is to define a model for evaluating the time to decompress a string $\mathcal{S}$ compressed via LZ77 in a hierarchical-memory setting. The decompression proceeds from left to right in $\mathcal{S}$ by reconstructing one phrase at a time. For each phrase $\langle d, \ell\rangle$, the decompressor needs to decode its codeword and then copy the substring of length ℓ at distance d from the current position in $\mathcal{S}$. In terms of memory accesses this means a random access to locate that copy plus the cost of reading it. Taking inspiration from models in [1, 2, 4, 28, 33], we assume that accessing a character at distance d takes $t(d)$ time, where $t(d) = \lceil \log d\rceil$, whereas scanning ℓ consecutive characters takes ℓ time regardless of the memory level containing these characters.[2]

Under these assumptions, the decompression of a phrase $\langle d, \ell\rangle$ takes $s(d, \ell)$ time to read and decode the codeword of d and ℓ, time $t(d) + \ell$ to read the copy, and time ℓ to append it to $\mathcal{S}$. Summing over all phrases, and recalling that their total length is n, we get a total decompression time of $t(\pi) = \sum_{\langle d,\ell\rangle\in\pi} 2\ell + \sum_{\langle d,\ell\rangle\in\pi}(t(d) + s(d, \ell)) = 2n + \sum_{\langle d,\ell\rangle\in\pi}(t(d) + s(d, \ell))$. Since the $2n$ term is independent of the parsing it

[2]See Drepper's monograph on memory hierarchies [10].

Table 1: Summary of main notations.

Name	Definition	Properties
$\mathcal{S}$	A (null-terminated) document to be compressed.	
n	Length of $\mathcal{S}$ (end-of-text character included).	
$\mathcal{S}[i]$	The i-th character of $\mathcal{S}$.	
$\mathcal{S}[i,j]$	Substring of $\mathcal{S}$ starting from $\mathcal{S}[i]$ until $\mathcal{S}[j]$	
$\langle 0, c\rangle$	A LZ77 phrase which represents a single character c.	
$\langle d, \ell\rangle$	A LZ77 phrase which represents a copy of a string of length ℓ at distance d	
$t(d)$	Amount of time spent in accessing the first character of a copy at distance d.	$t(d) = O(\log n)$.
$s(d,\ell)$	The length in bits of the encoding of $\langle d, \ell\rangle$.	$s(d,\ell) \leq s(d',\ell')$, for $d \leq d'$ and $\ell \leq \ell'$.
$t(d,\ell)$	The time needed to decompress the LZ77-phrase $\langle d, \ell\rangle$.	We have both $t(d,\ell) = t(d) + s(d,\ell)$ and $t(d,\ell) \leq t(d',\ell')$, for $d \leq d'$ and $\ell \leq \ell'$.
$s(\pi)$	The space occupancy of parsing π.	$s(\pi) = \sum_{\langle d,\ell\rangle \in \pi} s(d,\ell)$.
$t(\pi)$	The time needed to decompress the parsing π.	$t(\pi) = 2n + \sum_{\langle d,\ell\rangle \in \pi} t(d,\ell)$.
$s_{\max}$	The maximum space occupancy (in bits) of any LZ77 phrase of $\mathcal{S}$.	$s_{\max} = O(\log n)$ in our model.
$t_{\max}$	The maximum time taken to decompress a LZ77 phrase of $\mathcal{S}$.	$t_{\max} = O(\log n)$ in our model.
s_{costs}	The number of distinct values which may be assumed by $s(d,\ell)$ when $d \leq n$, $\ell \leq n$.	$s_{\text{costs}} = O(\log n)$ in our model.
t_{costs}	The number of distinct values which may be assumed by $t(d,\ell)$ when $d \leq n$, $\ell \leq n$.	$t_{\text{costs}} = O(\log n)$ in our model.

can be neglected, thus focusing on the terms $t(d,\ell) = t(d) + s(d,\ell)$ for each individual phrase of π. As in the previous section we denote by t_{costs} the number of distinct values which may be assumed by $t(d,\ell)$ when $d, \ell \leq n$; clearly $t_{\text{costs}} = O(\log n)$. Similarly to s_{costs}, this term will be crucial in defining the time complexity of our algorithm.

Pathological strings: space/time trade-offs matter. In our context we are interested in LZ77-parsings which "optimize" two criteria, namely *decompression time* and *compressed space*. In this respect, the notion of "best" parsing needs to recall the one of *Pareto-optimal* parsings, i.e., parsings which are not worse than some others in one parameter, being it the decompression time or the compressed space. Here we show, as claimed in the introduction, that there exists an infinite family of strings for which the Pareto-optimal parsings exhibit significant differences in their decompression time versus compressed space.

For the sake of presentation let us assume that each codeword takes constant space, and that our model of computation consists of just two memory levels such that the access time of the fastest level (of size c) is negligible, while the access time of the slowest level (of unbounded size) is substantial. We construct our pathological input string $\mathcal{S}$ as follows. Fix any string P of length at most c drawn over a alphabet Σ which can be LZ77-parsed with k phrases. For any $i \geq 0$, let B_i be the string $\$^{c+i}P$ with \$ a special symbol not in Σ. Our string $\mathcal{S}$ is $B_0 B_1 \ldots B_m$. Since the length of run of \$s increases as i increases, no pair of consecutive strings B_i and B_{i+1} can be part of the same LZ77-phrase. Moreover, we have two alternatives in parsing each B_i, with $i \geq 1$: (1) we parse B_i by deploying only its content and thus not requiring any cache miss at decompression time, this uses $2+k$ phrases which copy at distance at most c; (2) we parse B_i by using 2 phrases copied from the previous string B_{i-1}, thus requiring one cache miss at decompression time.

There are $m-1$ Pareto-optimal parsings of $\mathcal{S}$ obtained by choosing one of the above alternatives for each string B_i. On one extreme, the parser always chooses alternative (1) obtaining a parsing with $m(2+k)$ phrases which is decompressible with no cache misses. On the other extreme, the parser always prefers alternative (2) obtaining a parsing with $2+k+2m$ phrases which is decompressible with $m-1$ cache misses. In between these two extremes, we have a plethora of Pareto-optimal parsings: we can move from one extreme to the other by trading decompression speed for space occupancy. In particular, we can save k phrases at the cost of one more cache miss, where k is a value which can be varied by choosing different strings P. The ambitious goal of this paper is to automatically and efficiently choose any of these trade-offs.

3 From LZ77-Parsing to a weighted DAG

In this section we model the *bicriteria LZ77-parsing* problem as a *Weight-Constrained Shortest Path* problem (WCSPP) over a weighted DAG $\mathcal{G}$ defined as follows. Given an input string $\mathcal{S}$ of length[3] n,the graph $\mathcal{G}$ consists of n nodes, one per input character, and m edges, one per possible LZ77-phrase in $\mathcal{S}$. In particular we distinguish two types of edges: $(i, i+1)$, which represents the case of the single-character phrase $\langle 0, \mathcal{S}[i] \rangle$, and (i, j) with $j = i+\ell > i+1$, which represents the phrase $\langle d, \ell \rangle$ and thus the case of $\mathcal{S}[i, i+\ell-1]$ occurring d characters before in $\mathcal{S}$. This construction was proposed in [32]: clearly, $\mathcal{G}$ is a DAG and each path from node 1 to node n (1n-path) corresponds to an LZ77-parsing of $\mathcal{S}$. Subsequently, [17] added the weight $s(d, \ell)$ to the edge (i, j) in order to denote its space occupancy in bits.

We extend this modeling by adding another weight to $\mathcal{G}$'s edges, namely the time $t(i, j)$ taken to decode $\langle d, \ell \rangle$. This way, every 1n-path π not only identifies an LZ77-parsing of $\mathcal{S}$, but also the sum of the space-costs ($s(\pi)$) and the sum of the time-weights ($t(\pi)$) of its edges define its compressed bit-space occupancy and its decompression time, respectively. As a result of this modeling, we can re-phrase our bicriteria LZ77-parsing as the Weighted-Constrained Shorted Path problem in $\mathcal{G}$, which asks for $\min_{\pi \in \Pi} \; s(\pi)$ provided that $t(\pi) \leq T$. Clearly we could reverse the role of space and time in $\mathcal{G}$'s edges, but for ease of explanation, in the rest of the paper we will consider only the first formulation, even if our algorithmic solution can be used for both versions without any loss in its time/space efficiency.

In the following, we say that an edge (i', j') is *nested* in an edge (i, j) whenever $i \leq i' < j' \leq j$. To design efficient algorithms for WCSPP, it is crucial to exploit the peculiar properties of $\mathcal{G}$.

Property 3.1. *Given an edge (i, j) of $\mathcal{G}$, any (i', j') nested in (i, j) is (a) an edge of $\mathcal{G}$ and (b) its time- and space-weights are smaller or equal than the ones of (i, j).*

The first property derives from the fact that $\mathcal{G}$ models the parsing of a text using a prefix-/suffix-complete dictionary, as the LZ77 one. The second property derives from the fact that the functions $s(d, \ell)$ and $t(d, \ell)$, which model the time/space edge-weights, are non-decreasing in both arguments. So, given a phrase $\mathcal{S}[i, j]$ and its corresponding codeword $\langle d, \ell \rangle$, any substring $\mathcal{S}[i, j']$ is also a phrase (from the prefix-complete property) and its codeword $\langle d', \ell' \rangle$ is such that $d' \leq d$ and $\ell' \leq \ell$, because $\mathcal{S}[i, j']$ occurs at least wherever $\mathcal{S}[i, j]$ does.

[3] Recall that $\mathcal{S}$ is terminated by a special character.

3.1 Pruning the graph.

The size of $\mathcal{G}$ may be quadratic in n; just consider the string $\mathcal{S} = a^n$ which generates one edge per substring of $\mathcal{S}$. Given that n is typically of the order of millions or even billions, storing the whole $\mathcal{G}$ is unfeasible. This problem has been already faced in [17] while solving the bit-optimal LZ77-parsing problem over a graph with only the space-cost edges. Their solution mainly relied on two ideas: (i) pruning from their graph a large subset of unnecessary edges, yet guaranteeing that the bit-optimal path is preserved, and (ii) generating the forward stars of the nodes in the pruned graph on-the-fly by means of an algorithm, called FSG. It was shown in [17] that such pruned graph has size $O(n \, s_{\text{costs}})$ and can be generated incrementally in that time and only $O(n)$ space.

The contribution of this section is twofold: we show that there exists a small subgraph of $\mathcal{G}$, consisting of $O(n(s_{\text{costs}} + t_{\text{costs}}))$ edges, which includes all Pareto-optimal 1n-paths of $\mathcal{G}$; we then show that this pruned graph can be generated efficiently by using the FSG algorithm. The monotonicity property stated in Property 3.1 for the $s()$-cost and the $t()$-weight of DAG-edges allows us to define the notion of *maximality* of an edge, which (in turn) is correlated to the property of Pareto-optimality of a 1n-path in $\mathcal{G}$.

Definition 3.1. *An edge $e = (i, j)$ is said to be s-*maximal *iff, either the (next) edge $e' = (i, j+1)$ does not exist, or it does exist but the s-cost of e' is strictly larger than the s-cost of e. In a similar vein we define the notion of t-*maximal *edge, and state that an edge is* maximal *whenever it is either s-*maximal *or t-*maximal*, or both.*

Lemma 3.1 shows that, for any path π from node i to j and for each i' between i and j, there is a path from i' to j with cost/time not higher than those of π.

Lemma 3.1. *For each triple of nodes $i < i' < j$, and for each path π from i to j, there exists a path π' from i' to j such that $t(\pi') \leq t(\pi)$ and $s(\pi') \leq t(\pi)$.*

Proof. Let (h, k) be the edge of π which surpasses i' in $\mathcal{G}$, i.e, $h < i' \leq k$, and let π'' be the sub-path of π' from k to j. If $i' = k$, the thesis follows by setting

$\pi' = \pi''$, and noticing that this is a suffix subpath of π thus incurring in smaller costs. Otherwise, the edge (i', k) exists (because of the suffix-completeness property of LZ77-phrases), and its time and space weights are not greater than the corresponding ones of edge (h, k) (Property 3.1). Thus the thesis follows by setting $\pi' = (i', k) \cdot \pi''$.

The lemma stated above, used with $j = n$, allows to "push" to the right non-maximal edges by iteratively substituting non-maximal edges with maximal ones without augmenting the time and space costs of the path. This fact is exploited in Theorem 3.1, which shows that the search of optimal paths in $\mathcal{G}$ can be limited to those composed of maximal edges only.

THEOREM 3.1. *For any 1n-path π there exists a 1n-path $\pi^\star$ composed of maximal edges only and such that $\pi^\star$ is not worse than π in any one of its two costs, i.e., $t(\pi^\star) \leq t(\pi)$ and $s(\pi^\star) \leq s(\pi)$.*

Proof. We show that any 1n-path π containing non-maximal edges can be turned into a 1n-path π' containing maximal edges only. Take the leftmost non-maximal edge in π, say (v, w), and denote by π_v and π_w, respectively, the prefix/suffix of path π ending in v and starting from w. By definition of maximality, it must exist a maximal edge (v, z), with $z > w$, whose time/space weights are the same ones of (v, w). We can then apply Lemma 3.1 to the triple (w, z, n) and thus derive a path μ from z to n such that $s(\mu) \leq s(\pi_w)$ and $t(\mu) \leq t(\pi_w)$.

We then construct the 1n-path π'' by connecting the sub-path π_v, the maximal edge (v, z), and the path μ: using Lemma 3.1 one readily shows that the time/space costs of π'' are not larger than these of π. The key property is that we pushed right the leftmost non-maximal edge (if any), which must now occur (if ever) within μ; by iterating this argument we get the thesis.

Let $\widetilde{\mathcal{G}}$ be the pruned graph defined by keeping only maximal edges in $\mathcal{G}$. Since the set of maximal edges is given by the union of s-maximal and t-maximal edges, there cannot be more than $s_{\text{costs}} + t_{\text{costs}}$ maximal edges outgoing from any node. Given that both s_{costs} and t_{costs} are $O(\log n)$, it follows that $\widetilde{\mathcal{G}}$ has at most $O(n \log n)$ edges, and thus it is asymptotically sparser than $\mathcal{G}$. Due to lack of space we cannot dig into the generation of these edges (details in the journal paper), so here we state that all maximal edges of $\mathcal{G}$ can be generated on-the-fly by easily adapting the FSG-algorithm [17] and taking $O(1)$ amortized time per edge, hence overall $O(n \log n)$ time and $O(n)$ bits of working space. This means that the retrieval of the optimal path $\pi^\star$ can be done by examining only a (significantly smaller) sub-graph of $\mathcal{G}$ which can be generated in an optimal output-sensitive manner.

4 Our Approximation Algorithm

This section is devoted to solve WCSPP over the weighted DAG $\mathcal{G}$ whose structure and weights satisfy Property 3.1. Recall that $t_{\max}$ and $s_{\max}$ are, respectively, the maximum time-cost and the maximum space-weight of the edges in $\mathcal{G}$. We denote with $z(P)$ the optimal value of an optimization problem P, set $\varphi^\star = z(\texttt{WCSPP})$, and use WCSPP($\lambda$) to denote the Lagrangian relaxation of WCSPP with Lagrangian multiplier λ, namely:

$$(\texttt{WCSPP}\,(\lambda)) \qquad\qquad \min_{\pi \in \Pi}\; s(\pi) + \lambda(t(\pi) - T).$$

As mentioned in the introduction, our algorithm works in two phases. In the first phase, described in Section 4.1, the algorithm solves the Lagrangian Dual problem through a specialization of Kelley's cutting-plane algorithm [23], as first introduced by Handler and Zang [21]. The result is a lower-bound $z^\star$ for WCSPP and an instantiation for the parameter $\lambda^\star \geq 0$ which maximizes the optimal value of WCSPP(λ). In addition, this computes in almost linear time (Lemma 4.1) a pair of paths (π_L, π_R) which are optimal for WCSPP($\lambda^\star$) and are such that $t(\pi_L) \geq T$ and $t(\pi_R) \leq T$.

In case one path among them satisfies the time bound T exactly, then its space-cost equals the optimal value $\varphi^\star$, and thus that path is an optimal solution for WCSPP. Otherwise, the algorithm starts the second phase, described in Section 4.2, which is the more technical algorithmic contribution of this paper. This phase derives a new path by joining a proper prefix of π_L with a proper suffix of π_R. The key difficulty here is to show that this new path guarantees an additive-approximation of the optimal solution (Lemma 4.5), and it can be computed in just $O(n)$ time and $O(1)$ auxiliary space. At the end, we will have proved the following:

THEOREM 4.1. *There is an algorithm which computes a path π such that $s(\pi) \leq \varphi^\star + s_{\max}$ and $t(\pi) \leq T + 2\, t_{\max}$ in $O(n \log n \log(n\, t_{\max}\, s_{\max}))$ time and $O(n)$ space.*

We call this type of result an $(s_{\max}, 2\, t_{\max})$-additive approximation. By recalling that $s_{\max}$ and $t_{\max}$ are $O(\log n)$, since we are using universal integer encoders and memory hierarchies whose time access grows logarithmically (see Section 2), it holds:

COROLLARY 4.1. *There is an algorithm that computes an $(O(\log n), O(\log n))$-additive approximation of the Bicriteria data-compression problem in $O(n \log^2 n)$ time and $O(n)$ space.*

It is important to remark that this type of approximation is very strong because it is *additive* rather than *multiplicative* in the value of the bounded resources, as instead occur for the "classic" (α, β)-approximation [20]. In this additive-approximation, the absolute error stays constant as the value of the optimal solution grows, conversely to what occurs in the multiplicative-approximation for which, as the optimum grows, the absolute error grows too.

Interestingly, from Theorem 4.1 and our assumptions on $s_{\max}$ and $t_{\max}$ we can derive a FPTAS for our problem as stated in the following theorem (proof in the journal paper).

THEOREM 4.2. *For any fixed* $\epsilon > 0$*, then there exists a multiplicative* $\left(\epsilon, \frac{\epsilon}{2}\right)$*-approximation scheme for the Bicriteria data-compression problem which takes* $O\left(\frac{1}{\epsilon}\left(n \log^2 n + \frac{1}{\epsilon^2}\log^4 n\right)\right)$ *time and* $O(n + \frac{1}{\epsilon^3}\log^4 n)$ *space complexity.*

By setting $\epsilon > \sqrt[3]{\frac{\log^4 n}{n}}$, the bounds become $O\left(n \log^2 n / \epsilon\right)$ time and $O(n)$ space. Notice that both the FPTAS and the (α, β)-approximation guarantee to solve the Bicriteria data-compression problem in $o(n^2)$ time complexity, which was our original goal.

4.1 First phase: The cutting-plane algorithm.

The first phase consists of solving the Lagrangian dual of problem `WCSPP` through the first phase of Handler and Zang's seminal paper [21]. Our key observation is that each iteration can be implemented by solving a bit-optimal `LZ77`-problem formulated over the pruned graph $\widetilde{\mathcal{G}}$.

The Lagrangian dual of problem `WCSPP` is $\max_{\lambda \geq 0} \min_{\pi \in \Pi} s(\pi) + \lambda(t(\pi) - T)$. This can be rewritten as a (very large) linear program in which every 1n-path defines one of the constraints and, possibly, one face of the feasible region: $\max_{\lambda \geq 0} \{ u : u \leq s(\pi) + \lambda(t(\pi) - T), \forall \pi \in \Pi \}$.

This can be interpreted geometrically. Let us denote as $L(\pi, \lambda)$, or λ-*cost*, the Lagrangian cost $s(\pi) + \lambda(t(\pi) - T)$ of the path π with parameter λ. Each path π represents thus the line $\varphi = L(\pi, \lambda)$ in the Euclidian space (λ, φ). Feasible paths have a non-positive slope (since $t(\pi) \leq T$), unfeasible paths have a positive slope (since $t(\pi) > T$). Let us now consider the Lagrangian function $\varphi(\lambda) = \min_{\pi \in \Pi} L(\pi, \lambda)$. This function is piecewise linear and represents the lower envelope of all the "lines" in Π. A convenient way of interpreting the large linear program above is as the problem of maximizing the function $\varphi(\lambda)$ over all $\lambda \geq 0$. Unfortunately, the exponential number of paths makes impossible to solve this by a brute-force approach. However, the full set of paths Π is not needed. In fact, we can use a *cutting-plane* method [23] which determines a pair of paths (π_L, π_R) such that (i) $L(\pi_L, \lambda^\star) = L(\pi_R, \lambda^\star)$ = the optimal (maximum) value of $\varphi(\lambda)$ and (ii) $t(\pi_L) \geq T$ and $t(\pi_R) \leq T$. Referring to the terminology often used to describe the simplex method [31], these paths correspond to a (feasible) *optimal basis* of the linear program.

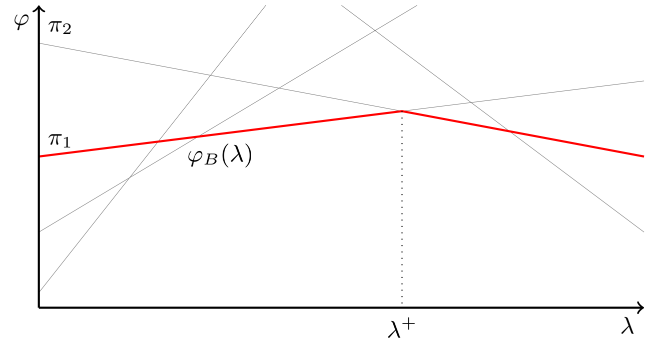

Figure 1: Each path $\pi \in B$ is a line $\varphi = L(\pi, \lambda)$, and $\varphi_B(\lambda)$ (in red) is given by the lower envelope of lines π_1 and π_2. Notice that the maximum value of function φ_B is the same as the maximum value of the lower envelope of *all* the lines in the space.

The cutting-plane method is iterative and best explained geometrically. At each step, the algorithm generates one of the paths in Π and keeps a set B as formed by *two* 1n-paths, denoted as π_1 and π_2 through this section. Set B defines the restricted Lagrangian function $\varphi_B(\lambda)$ which is a restriction of the function $\varphi(\lambda)$ to the paths $B \subseteq \Pi$, as illustrated in Figure 1. The algorithm maintains the following invariant: the maximum of function $\varphi_B(\lambda)$, which is given by the intersection of π_1 and π_2, is equal to the maximum value of the function defined by the lower envelope of *all* the lines defined by the paths generated thus far. At the beginning B is given by the *space-optimal* and the *time-optimal* paths, which can be obtained by means of two shortest path computations over $\widetilde{\mathcal{G}}$. It is easy to see that π_1 must have a non-negative slope (thus $t(\pi_1) \geq T$) and π_2 must have a non-positive slope ($t(\pi_2) \leq T$) for otherwise one of the two is optimal and we can stop; this invariant is kept true along the iterations. It is also easy to see that the intersection point λ^+ between π_1 and π_2 correspond to the maximum of the function $\varphi_B(\lambda)$, as illustrated in Figure 1. Since $\varphi(\lambda)$ may be interpreted as the lower envelope of a set of lines given by paths in $\Pi \supseteq B$, it holds $\varphi_B(\lambda) \geq \varphi(\lambda)$ for each $\lambda \geq 0$. As a corollary $\varphi_B(\lambda^+) \geq \varphi(\lambda^\star)$, i.e., the optimal value of $\varphi_B(\lambda)$ is an upper-bound to the optimal value of $\varphi(\lambda)$. In particular, $\varphi_B(\lambda^+)$ is strictly greater than $\varphi(\lambda^\star)$ when B is not an optimal basis.

At each step, the algorithm knows the value λ^+

(by induction) which maximizes $\varphi_B(\lambda)$, for the current subset B. Then it computes a path π^+ for which $L(\pi^+, \lambda^+) = \varphi(\lambda^+) = \min_{\pi \in \Pi} L(\pi, \lambda^+)$ (according to definition of φ). Our key observation here is that path π^+ can be determined by searching for a shortest path whose (relaxed) cost is evaluated as $s(\pi) + \lambda^+ t(\pi)$ within the pruned DAG $\widetilde{\mathcal{G}}$. Nicely, this search can be implemented via an adaptation of the FSG-algorithm [17] thus taking $O(n \log n)$ time and $O(n)$ space, as already remarked above.

In the case that the computed $\varphi(\lambda^+) = L(\pi^+, \lambda^+)$ equals $\varphi_B(\lambda^+)$ (which is known by induction) then the pair (π_1, π_2) is an optimal basis, and the algorithm stops by setting $\lambda^\star = \lambda^+$ and $(\pi_L, \pi_R) = (\pi_1, \pi_2)$. Otherwise, the algorithm must update B to maintain the invariant on $\varphi_B(\lambda)$ stated above. A simple geometric argument shows B can be updated as (π_1, π^+) if π^+ if feasible and as (π^+, π_2) if it is not. Moreover, the algorithm updates λ^+ to reflect the new optimal value of φ_B.

The last question is for how many iterations we have to run the cutting-plane algorithm above. Mehlhorn and Ziegelmann have shown [30] that, for the case where the costs and the resources of each arc are integers belonging to the compact sets $[0, C]$ and $[0, R]$ respectively, then the cutting-plane algorithm (which they refer to as the *Hull approach*) terminates in $O(\log(nRC))$ iterations. In our context $R = C = O(n)$:

Lemma 4.1. *The first phase computes a lower-bound $z^\star$ for* WCSPP*, an instantiation for $\lambda^\star \geq 0$ which maximizes the optimal value of* WCSPP(λ)*, and a pair of paths (π_L, π_R) which are optimal for* WCSPP($\lambda^\star$)*. This takes $O(\tilde{m} \log(n\ t_{\max}\ s_{\max}))$ time and $O(n)$ space, where $\tilde{m} = O(n \log n)$ is $\widetilde{\mathcal{G}}$'s size.*

4.2 Second phase: The path-swapping algorithm. Unfortunately, it is not easy to bound the solution computed with Lemma 4.1 in terms of the space-optimal solution of WCSPP. Therefore the second phase of our algorithm is the technical milestone that allows to turn the basis (π_L, π_R) into a path whose time- and space-costs can be mathematically bounded in terms of the optimal solution for WCSPP. In the following we denote a path as a sequence of increasing node-IDs and do not allow a node to appear multiple times in a path, so a path (v, w, w, w, z) must be intended as (v, w, z). Moreover, we use the following notation.

- $\mathsf{Pref}(\pi, v)$ is the *prefix* of a 1n-path π ending into the largest node $v' \leq v$ in π.

- $\mathsf{Suf}(\pi, v)$ is the *suffix* of a 1n-path π starting from the smallest node $v'' \geq v$ in π.

Given two paths π_1 and π_2 in $\mathcal{G}$, we call *path swapping* through a *swapping-point* v, which belongs either to π_1 or π_2 (or both), the operation which creates a new path, denoted by $\mathrm{ps}(\pi_1, \pi_2, v) = (\mathsf{Pref}(\pi_1, v), v, \mathsf{Suf}(\pi_2, v))$, that connects a prefix of π_1 with a suffix of π_2 via v.

Property 3.1 guarantees that the path-swap operation is well-defined and, in fact, the next Fact 4.1 states that we always have edges to connect the last node of $\mathsf{Pref}(\pi_1, v)$ with v, and v with the first node of $\mathsf{Suf}(\pi_2, v)$. An illustrative example is provided in Figure 2.

Fact 4.1. *The path-swap operation is well-defined for each pair of 1n-paths (π_1, π_2) and for each swapping-point v which belongs either to π_1 or π_2 (or both).*

For any given $\lambda \geq 0$, a path π is λ-*optimal* if its Lagrangian cost $L(\pi, \lambda)$ is equal to the value of the Lagrangian function $\varphi(\lambda)$. The following lemma shows that any path-swap of two λ-optimal paths is off at most $t_{\max}$ in time and $s_{\max}$ in space from being a λ-optimal path.

Lemma 4.2. *Let π_1, π_2 be λ-optimal paths, for some $\lambda \geq 0$. Consider the path $\pi_A = \mathrm{ps}(\pi_1, \pi_2, v)$, where v is an arbitrary swapping point. There exist values s, t such that $s \leq s(\pi_A) \leq s + s_{\max}$, $t \leq t(\pi_A) \leq t + t_{\max}$ and $s + \lambda(t - T) = \varphi(\lambda)$.*

Proof. Let $\pi_B = \mathrm{ps}(\pi_2, \pi_1, v)$: we claim that

$$L(\pi_A, \lambda) + L(\pi_B, \lambda) \leq 2\varphi(\lambda) + s_{\max} + \lambda\, t_{\max}$$

which then immediately gives the thesis since $\varphi(\lambda) \leq L(\pi, \lambda)$ for each 1n-path π.

Let us denote by $\ell(i, j)$ the scalarized cost $s(i, j) + \lambda\, t(i, j)$ of edge (i, j), and by $\ell(\pi) = \sum_{(i,j) \in \pi} \ell(i, j)$ the sum of the scalarized costs of all edges in π, so that $L(\pi, \lambda) = \ell(\pi) - \lambda T$. Moreover, let us use the notation $P_j = \ell(\mathsf{Pref}(\pi_j, v))$ and $S_j = \ell(\mathsf{Suf}(\pi_j, v))$ for, respectively, the scalarized costs of the prefix and suffix of the path π_j before/after the swapping point v. There are three cases to consider:

1. *v belongs to both π_1 and π_2*: In this case, we have $\ell(\pi_A) = P_1 + S_2$, $\ell(\pi_B) = P_2 + S_1$, $\ell(\pi_1) = P_1 + S_1$ and $\ell(\pi_2) = P_2 + S_2$. Since $\ell(\pi_1) + \ell(\pi_2) = \ell(\pi_A) + \ell(\pi_B)$ and π_1 and π_2 are λ-optimal paths, we have $L(\pi_A, \lambda) + L(\pi_B, \lambda) = L(\pi_1, \lambda) + L(\pi_2, \lambda) = 2\varphi(\lambda)$ from which our claim follows (with equality).

2. *v does not belong to π_1*: let v' and v'' be, respectively, the rightmost node preceding v and the leftmost node following v in π_1 (see Figure 2). We have

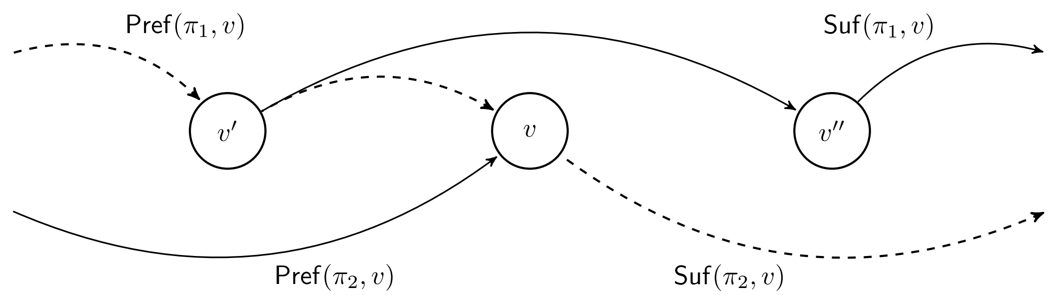

Figure 2: A path-swap of π_1, π_2 at the swapping point v. The resulting path is dashed.

- $\ell(\pi_1) = P_1 + \ell(v', v'') + S_1$;
- $\ell(\pi_2) = P_2 + S_2$;
- $\ell(\pi_A) = P_1 + \ell(v', v) + S_2$;
- $\ell(\pi_B) = P_2 + \ell(v, v'') + S_1$.

By using the above relations we have

$$\ell(\pi_A) + \ell(\pi_B) =$$
$$P_1 + \ell(v', v) + S_2 + P_2 + \ell(v, v'') + S_1 =$$
$$\ell(\pi_1) + \ell(\pi_2) - \ell(v', v'') + \ell(v', v) + \ell(v, v'')$$

which then gives our claim observing that π_1 and π_2 are λ-optimal paths, $\ell(v', v) \leq \ell(v', v'')$ due to the non-decreasing cost property, and $\ell(v, v'') \leq s_{\max} + \lambda t_{\max}$.

3. *v does not belong to π_2*: this case is symmetric to the previous one.

Now, consider two paths π_1, π_2 to be swapped and two *consecutive swapping points*, that is, two nodes v and w belonging to either π_1 or π_2 and such that there is no node z belonging to π_1 or π_2 with $v < z < w$. The lemma below states that time and space of paths $\mathrm{ps}(\pi_1, \pi_2, v)$ and $\mathrm{ps}(\pi_1, \pi_2, w)$ differ by at most $t_{\max}$ and $s_{\max}$.

Lemma 4.3. *Let π_1, π_2 be two paths to be swapped. Let also v and w be two consecutive swapping points. Set $\pi = \mathrm{ps}(\pi_1, \pi_2, v)$ and $\pi' = \mathrm{ps}(\pi_1, \pi_2, w)$: then, $|s(\pi) - s(\pi')| \leq s_{\max}$ and $|t(\pi) - t(\pi')| \leq t_{\max}$.*

Proof. Let us consider the sub-paths $\mathsf{Pref} = \mathsf{Pref}(\pi, v)$ and $\mathsf{Pref}' = \mathsf{Pref}(\pi', w)$. There are two cases:

1. $v \in \pi_1$: in this case, $\mathsf{Pref}' = (\mathsf{Pref}, w)$. Thus, $s(\mathsf{Pref}') - s(\mathsf{Pref}) = s(v, w)$ and $t(\mathsf{Pref}') - t(\mathsf{Pref}) = t(v, w)$;

2. $v \notin \pi_1$: let $\mathsf{Pref} = (v_1, \ldots, v_k, v)$; in this case, we have $\mathsf{Pref}' = (v_1, \ldots, v_k, w)$. Thus, we have $s(\mathsf{Pref}') - s(\mathsf{Pref}) = s(v_k, w) - s(v_k, v) \leq s_{\max}$; a similar argument holds for the time weight.

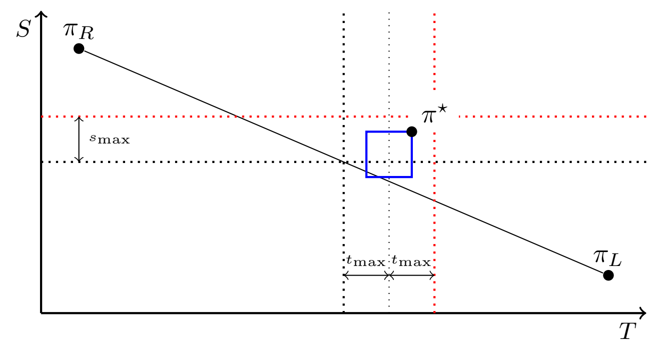

Figure 3: Geometrical interpretation of Lemmas 4.2 and 4.3. Paths are represented as points in the time-space coordinates. Path $\pi^\star$ is obtained by path-swapping paths π_L and π_R. The blue rectangle is guaranteed by Lemma 4.2 to intersect with the segment from π_L to π_R, while Lemma 4.3 guarantees that there is at least one path-swapped solution having time coordinates between t and $t + t_{\max}$ for any $t \in [t(\pi_R), t(\pi_L)]$, in this case $[T + t_{\max}, T + 2t_{\max}]$.

Thus, $s(\mathsf{Pref}') - s(\mathsf{Pref}) \leq s_{\max}$ and $t(\mathsf{Pref}') - t(\mathsf{Pref}) \leq t_{\max}$. Symmetrically, it holds $s(\mathsf{Suf}) - s(\mathsf{Suf}') \leq s_{\max}$ and $t(\mathsf{Suf}) - t(\mathsf{Suf}') \leq t_{\max}$; since $s(\pi) = s(\mathsf{Pref}) + s(\mathsf{Suf})$ and $s(\pi') = s(\mathsf{Pref}') + s(\mathsf{Suf}')$, it follows $|s(\pi) - s(\pi')| \leq s_{\max}$, and a similar argument holds for $|t(\pi) - t(\pi')|$.

Figure 3 gives a geometrical interpretation of this lemmas and shows, in an intuitive way, that it is possible to path-swap the optimal basis (π_L, π_R) computed by the cutting-plane algorithm (Lemma 4.1) to get an additive $(s_{\max}, 2\, t_{\max})$-approximation to the `WCSPP` by carefully picking a swapping point v. This result is deployed to prove the following.

Lemma 4.4. *Given an optimal basis (π_L, π_R) with $t(\pi_L) > T$ and $t(\pi_R) < T$, there exists a swapping point $v^\star$ and a path-swapped path $\pi^\star = \mathrm{ps}(\pi_1, \pi_2, v^\star)$ such that $t(\pi^\star) \leq T + 2\, t_{\max}$ and $s(\pi^\star) \leq \varphi^\star + s_{\max}$.*

Proof. Since $\mathrm{ps}(\pi_L, \pi_R, v_1) = \pi_R$ and $\mathrm{ps}(\pi_L, \pi_R, v_n) = \pi_L$, Lemma 4.3 implies that there must exist some $v^\star$ such that the path $\pi^\star = \mathrm{ps}(\pi_L, \pi_R, v^\star)$ has time $t(\pi^\star) \in [T + t_{\max}\,, T + 2\, t_{\max}]$. Due to Lemma 4.2, there are $s \geq s(\pi^\star) - s_{\max}$ and $t \geq T$ (since $t + t_{\max} \geq$

$t(\pi^\star) \geq T + t_{\max}$) such that $s + \lambda(t - T) = \varphi^\star$; hence $s \leq \varphi^\star$, which ultimately yields that $s(\pi^\star) \leq \varphi^\star + s_{\max}$.

The gap-closing procedure consists thus on choosing the best path-swap of the optimal basis (π_L, π_R) with time-weight within $T + 2\,t_{\max}$. The solution can be selected by scanning left-to-right all the swapping points, and evaluating the time cost and space weight for each candidate. This procedure can be implemented by keeping the time and space of the current prefix of π_L and suffix of π_R, and by updating them every time a new swapping point is considered. Since each update can be performed in $O(1)$ time, we obtain the following lemma, which combined with Lemma 4.1, proves our main Theorem 4.1.

LEMMA 4.5. *Given an optimal basis (π_L, π_R) of problem D', an additive $(s_{\max}, 2\,t_{\max})$-approximation to* `WCSPP` *can be found in $O(n)$ time and $O(1)$ auxiliary space.*

5 Experimental results

We describe here the preliminary results we obtained by executing `BC-ZIP`, an in-memory `C++` implementation of our `LZ77`-based data-compression scheme introduced in this paper. These experiments aim not only at establishing the ultimate performance of our compressor, but also at investigating the following three issues:

1) Trade-off range In Section 3 we motivated the interest in the Time-Constrained Space-Optimal `LZ77`-Parsing problem by showing a series of pathological texts for which the `LZ77`-parsings exhibit wide space-time trade-offs. In this section, we provide experimental evidence that these pathological cases do occur in practice, so that the design of a flexible compressor, as the one we propose in this paper, is worth not only in theory.

2) Estimating compression ratio The number of phrases is a popular metric for estimating the compression ratio induced by a `LZ77`-parsing. Ferragina et al. showed [17] that this is a simplistic metric, since there is a $\Omega\left(\frac{\log \log n}{\log n}\right)$ multiplicative gap in the compressed-space achieve by the bit-optimal parsing and the greedy one. In this section we deepen this argument by comparing experimentally the time-space trade-off when compressed space is either estimated exactly or approximated by the number of phrases in the parsing. The net result is to show that the number-of-phrases is a bad estimate for the space-occupancy of a `LZ77`-based compressor, so the space-time trade-offs obtained by algorithms based only on this measure can be widely off-mark of the true ones.

3) Comparing to the state-of-the-art Our experiments are executed against many state-of-the-art compression libraries. We executed the experiments over datasets of several types of data: ***Wikipedia*** (natural language), *DBLP* (XML), *PFAM* (biological data, [13]), and ***U.S. Census*** (database). Each dataset, available at `http://acube.di.unipi.it/bc-zip-dataset/`, consists of a chunk of 1GiB (2^{30} bytes). We compared our compressor `BC-ZIP` against the most popular and top-performing compressors belonging to the two main families: `LZ77`-based and BWT-based. From the former family, we included: (i) ***zlib*** which is the core of the well-known `gzip` compressor; (ii) *LZMA2* which is the core of `7zip` compressor and is appreciated for its high compression ratio and competitive decompression speed. From the latter family, we included: (i) ***bzip2*** which is a general purpose compressor available on any Linux distributions; and (ii) *BWT-Booster* which is the state-of-the-art for BWT-based compressors [15]. Moreover, we included ***Snappy*** and `LZ4` which are highly engineered `LZ77`-compressors used in BigTable [7] and Hadoop, offering a very fast decompression speed.

Each decompressor has been implemented in `C++`, to work in-memory, and it has been compiled with `g++` version `4.6.3` with options `-O3 -fomit-frame-pointer -march=native` and evaluated on a machine with the following characteristics: (i) processor: Intel Core 2 Duo P8600, with 64k of `L1` cache and 3mb of `L2` cache; (ii) RAM: 4GB `DDR3 PC3-8500`; (iii) Operating system: Ubuntu 12.04.

On compression we used the parameters yielding the best compression ratios, when available. In particular, results in Figure 5 have been obtained by running `bzip -9`, `gzip -9`, `lz4 -9` and `xz -9` (for `LZMA`) while compressing the dataset. We notice that both `Snappy` and `BWT-Booster` offer just one fixed trade-off, thus not offering any compression parameter to tune.

Implementation details. In implementing `BC-ZIP` we resorted to a simple byte-oriented encoder for the `LZ77`-phrases which alleviates the detrimental effects of branching codes. Encoding a phrase requires at most 2 bytes for the length and 4 bytes for the distance (so $s_{\max} = 48$ bits). Then we modeled the time-costs of the edges by three values which have been determined through many benchmarks (details in the journal paper), and we got $t_{\max} \approx 0.125\mu s$.

In order to create the graph $\mathcal{G}$, we developed two ways of assigning time/space weights to edges: the *full* model and the *fixed* model. Those models differ in the way they assign the space weight to edges. In the "full" model, the space weight of an edge is a measure of the bit-length of the corresponding codeword, while in the "fixed" model each edge has unitary weight. Thus, the space weight of a parsing in the "full" model is its compressed size, while in the "fixed" model is its number of phrases. Both models assign time weight to edges according to the time model illustrated in Section 2, where parameters have been derived experimentally. In particular, the time weight of a phrase $\langle d, \ell \rangle$ is t_1 if $d < 16000$, t_2 if $d \in [16000, 2300000]$, and t_3 otherwise, where the parameters t_i are derived by executing a proper benchmark over our machine (details in the journal paper).

At compression time, the user can specify a *time bound* T (in millisecs) or a *compression level* $C = (T - T_t)/(T_s - T_t)$, where T_t is the decompression time of the time-optimal parsing and T_s is the decompression time of the most succinct space-optimal parsing. We notice that compression level $C = 0$ (resp. $C = 1$) corresponds to the parsing with fastest decompression time (resp. smallest compressed space).

Experimental trade-off spectrum. We observed the experimental shape of the time-space trade-offs curve by compressing each dataset with linearly varying compression level from 0.2 to 1, considering both the fixed and the full model. Figure 4 shows that, for the full model, the space-time trade-off curve of the linearly changing compression level is actually linear too, which clearly shows that the trade-off can be effectively controlled in a principled way by our compressor.

Results are instead far less significant for the fixed model, in which space is estimated as the number of phrases in the parsing. Even if its curve is close to the one of the full model for `DBLP`, the curves are significantly different for the other three datasets. Moreover, the space-optimal parsing generated in the fixed-model with compression level 1 (which is equivalent to the greedy parsing) is dominated by the parsings generated with compression levels from 0.7 to 0.9 in `U.S. Census`, while parsings with compression level 0.7 and 0.8 are dominated by parsings with compression level 0.9 and 1 in `Wikipedia`. This clearly shows that the number of phrases is a poor metric for estimating the compression ratio of a parsing, and it offers a very simplistic estimate of the decompression time.

Comparison with state-of-the-art. Figure 5 reports the performance of the various compression algorithms on the datasets. Results show that the performance of our `BC-ZIP` are extremely good. On the one hand, it generates parsings with decompression time better than those of `LZ4` in three out of four datasets (`DBLP`, `PFAM`, `U.S. Census`), whereas for the fourth dataset (`Wikipedia`) `BC-ZIP` achieves a decompression time which is a little bit worse than `LZ4` but with a significantly improved compression ratio. On the other hand, its compression ratio at higher compression levels is close to the best one, namely that of `LZMA2` (excluding `BWT-Booster`, which exhibit an exceedingly slow decompression time), but with an order of magnitude faster decompression speed. Compression ratios of `BC-ZIP` are indeed very remarkable, because it uses a very simple byte-oriented encoder opposed to the statistical Markov-Chain encoder used in `LZMA2`.

Overall, these results show that not only our approach allows to effectively control the time-space trade-off in a practical yet principled manner; by explicitly taking into account both decompression-time and compressed-space, `BC-ZIP` leads to parsings which are *faster to decode and more space-succinct* than those generated by highly tuned and engineered parsing *heuristics*, like those of `Snappy` and `LZ4`.

6 Conclusions

We conclude this paper by mentioning two interesting future directions where the novel optimization-based approach proposed in this paper could be tested. The first one concerns the practical impact of these techniques on real big-data applications and their storage systems, like Hadoop. The second question, more of a theoretical vein, is whether it is possible to extend this novel bicriteria optimization approach to other interesting compressor families such as `PPM` and `BWT`.

References

[1] Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. In *ACM STOC*, pages 305–314, 1987.

[2] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Hierarchical memory with block transfer. In *IEEE FOCS*, pages 204–216, 1987.

[3] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.

[4] Bowen Alpern, Larry Carter, Ephraim Feig, and Ted Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2/3):72–109, 1994.

[5] Dhruba Borthakur and *et alii* Jonathan Gray. Apache hadoop goes realtime at facebook. In *ACM SIGMOD Conference*, pages 1071–1080, 2011.

[6] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital, 1994.

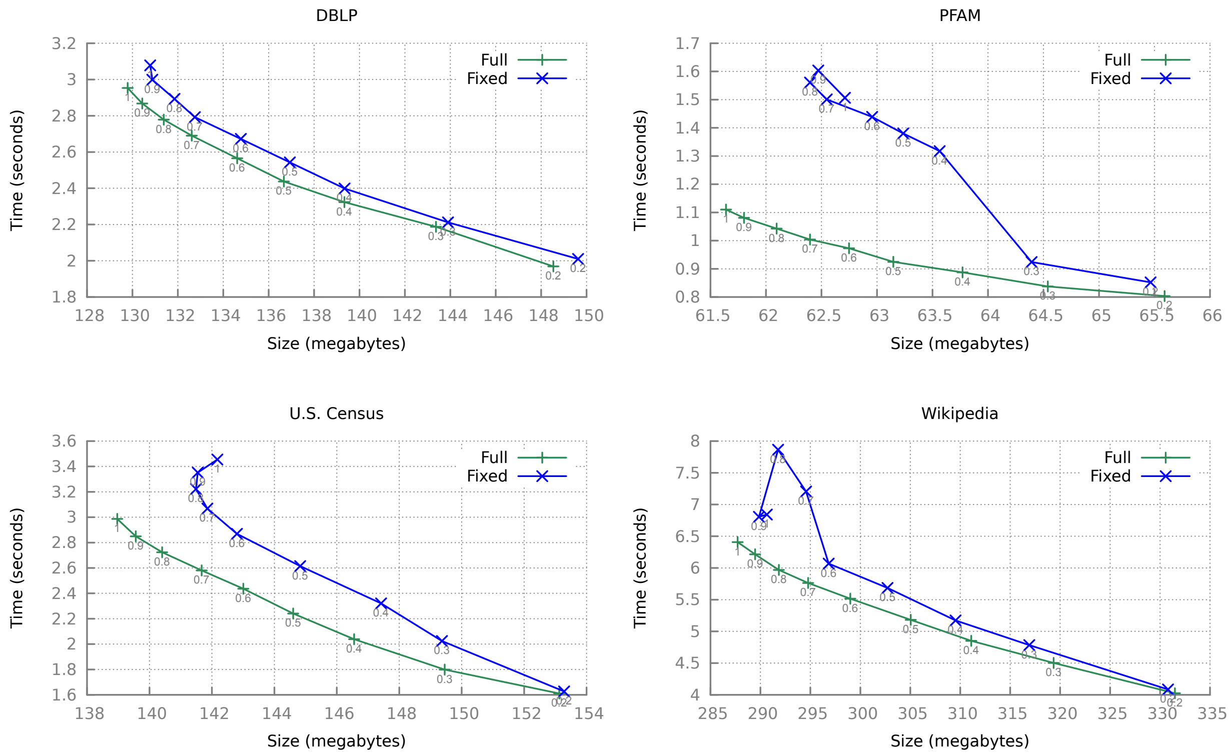

Figure 4: Decompression time and compressed space trade-offs obtained by changing the compression level from 0.2 to 1. The fractional numbers close to each curve specify the compression level achieved by the parsing providing that space/time trade-off. In the "full" model (green line), the space cost of a codeword is given by its length, while in the "fixed" model the space cost is unitary. In this plot, 1MB is 10^6 bytes.

Dataset	Parsing	Compressed size (MB, 10^6 bytes)	Decompression time (seconds)	Dataset	Parsing	Compressed size (MiB, 10^6 bytes)	Decompression time (seconds)
DBLP	BC-ZIP - 1	129.8	2.95	U.S. Census	BC-ZIP - 1	139.0	2.98
	BC-ZIP - 0.9	130.4	2.86		BC-ZIP - 0.9	139.6	2.84
	BC-ZIP - 0.8	131.4	2.77		BC-ZIP - 0.8	140.4	2.72
	BC-ZIP - 0.7	132.6	2.69		BC-ZIP - 0.7	141.7	2.58
	BC-ZIP - 0.6	134.6	2.56		BC-ZIP - 0.6	143.0	2.43
	BC-ZIP - 0.5	136.7	2.43		BC-ZIP - 0.5	144.6	2.24
	BC-ZIP - 0.4	139.3	2.32		**BC-ZIP - 0.4**	**146.6**	**2.03**
	BC-ZIP - 0.3	143.4	2.18		BC-ZIP - 0.3	149.5	1.79
	BC-ZIP - 0.2	**148.5**	**1.96**		BC-ZIP - 0.2	153.1	1.61
	Snappy	323.4	2.13		Snappy	324.1	2.28
	LZ4	214.7	1.98		LZ4	225.0	2.01
	zlib	191.6	11.65		zlib	176.3	11.44
	LZMA2	120.4	20.47		LZMA2	124.3	20.34
	bzip2	121.4	48.98		bzip2	180.7	50.40
	BWT-Booster	98.2	> 100		BWT-Booster	141.9	> 100
PFAM	**BC-ZIP - 1**	**61.6**	**1.11**	Wikipedia	BC-ZIP - 1	287.7	6.40
	BC-ZIP - 0.9	61.8	1.08		BC-ZIP - 0.9	289.5	6.21
	BC-ZIP - 0.8	62.1	1.04		BC-ZIP - 0.8	291.8	5.96
	BC-ZIP - 0.7	62.4	1.00		BC-ZIP - 0.7	294.8	5.76
	BC-ZIP - 0.6	62.7	0.97		BC-ZIP - 0.6	299.0	5.51
	BC-ZIP - 0.5	63.1	0.92		BC-ZIP - 0.5	305.0	5.18
	BC-ZIP - 0.4	63.8	0.88		BC-ZIP - 0.4	311.1	4.85
	BC-ZIP - 0.3	64.5	0.83		BC-ZIP - 0.3	319.3	4.50
	BC-ZIP - 0.2	65.6	0.80		**BC-ZIP - 0.2**	**331.5**	**4.02**
	Snappy	147.6	1.70		Snappy	585.7	2.84
	LZ4	74.4	1.41		LZ4	435.1	2.63
	zlib	62.3	7.63		zlib	380.3	17.63
	LZMA2	39.1	7.16		LZMA2	253.9	39.09
	bzip2	48.7	21.65		bzip2	304.5	66.64
	BWT-Booster	54.7	> 100		BWT-Booster	228.8	> 100

Figure 5: Rows "BC-ZIP - c" stands for the performance of our implementation of the Time-Constrained Space-Optimal LZ77 parsing with compression level c.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[8] Z. Cohen, Yossi Matias, S. Muthukrishnan, Süleyman Cenk Sahinalp, and Jacob Ziv. On the temporal hzy compression scheme. In *ACM-SIAM SODA*, pages 185–186, 2000.

[9] Graham Cormode and S. Muthukrishnan. Substring compression problems. In *ACM-SIAM SODA*, pages 321–330, 2005.

[10] Ulrich Drepper. What every programmer should know about memory. `http://www.akkadia.org/drepper/cpumemory.pdf`, 2007.

[11] Irina Dumitrescu and Natashia Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153, 2003.

[12] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

[13] Marco Punta *et al.* The pfam protein families database. *Nucleic Acids Research*, 40(Database-Issue):290–301, 2012.

[14] Martin Farach and Mikkel Thorup. String matching in Lempel-Ziv compressed strings. *Algorithmica*, 20(4):388–404, 1998.

[15] Paolo Ferragina, Raffaele Giancarlo, Giovanni Manzini, and Marinella Sciortino. Boosting textual compression in optimal linear time. *Journal of the ACM*, 52(4):688–713, 2005.

[16] P. Ferragina, I. Nitto, R. Venturini. On optimally partitioning a text to improve its compression. *Algorithmica: Special issue on ESA09's selected papers*, 61(1): 51-74, 2011.

[17] Paolo Ferragina, Igor Nitto, and Rossano Venturini. On the bit-complexity of Lempel-Ziv compression. In *ACM-SIAM SODA*, pages 768–777, 2009.

[18] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[19] Solomon Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[20] Errol L. Lloyd and S. S. Ravi. Topology control problems for wireless ad hoc networks. In *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*, pages 67–1 — 67–20. Chapman & Hall/CRC, 2007.

[21] Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[22] Orgad Keller, Tsvi Kopelowitz, Shir Landau, and Moshe Lewenstein. Generalized substring compression. In *CPM*, LNCS 5577, pages 26–38, Springer, 2009.

[23] J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[24] Dominik Kempa and Simon J. Puglisi. Lempel-Ziv factorization: fast, simple, practical. In *SIAM ALENEX*, pages 103–112, 2013.

[25] S. Rao Kosaraju and Giovanni Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999.

[26] Sebastian Kreft and Gonzalo Navarro. Lz77-like compression with fast random access. In *DCC*, pages 239–248, 2010.

[27] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Books on Mathematics Series. Dover Publ., 2001.

[28] Fabrizio Luccio and Linda Pagli. A model of sequential computation with pipelines access to memory. *Mathematical Systems Theory*, 26(4):343–356, 1993.

[29] Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Bicriteria network design problems. *J. Algorithms*, 28(1):142–171, 1998.

[30] Kurt Mehlhorn and Mark Ziegelmann. Resource constrained shortest paths. In *ESA*, LNCS 1879, pages 326–337, Springer, 2000.

[31] John A. Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[32] Ernst J. Schuegraf and H. S. Heaps. A comparison of algorithms for data base compression by use of fragments as language elements. *Information Storage and Retrieval*, 10(9-10):309–319, 1974.

[33] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory II: Hierarchical multilevel memories. *Algorithmica*, 12(2/3):148–169, 1994.

[34] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, 2. edition, 1999.

[35] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[36] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.