Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Search Strategies for Optimal Multi-Way Number Partitioning

Michael D. Moffitt
IBM Corp.
mdmoffitt@us.ibm.com

Abstract

The number partitioning problem seeks to divide a set of n numbers across k distinct subsets so as to minimize the sum of the largest partition. In this work, we develop a new optimal algorithm for multi-way number partitioning. A critical observation motivating our methodology is that a globally optimal k-way partition may be recursively constructed by obtaining *suboptimal* solutions to subproblems of size $k-1$. We introduce a new principle of optimality that provides necessary and sufficient conditions for this construction, and use it to strengthen the relationship between sequential decompositions by enforcing upper and lower bounds on intermediate solutions. We also demonstrate how to further prune unpromising partial assignments by detecting and eliminating dominated solutions. Our approach outperforms the previous state-of-the-art by up to *four orders of magnitude*, reducing average runtime on the largest benchmarks from several hours to less than a second.

1 Introduction

Number partitioning is one of the simplest problems in combinational optimization: it seeks to divide n positive integers across k mutually exclusive and collectively exhaustive subsets $\langle S_1, S_2, ..., S_k\rangle$ so as to minimize the sum of the largest set. For instance, given the integers $\{1,2,3,4,5\}$, the 3-way partitioning $\langle\{1,4\},\{2,3\},\{5\}\rangle$ is an optimal solution that perfectly balances all subsets to an equal sum of 5. Despite its simplicity, number partitioning is nevertheless NP-complete [Garey and Johnson, 1979] and often used to prove the NP-completeness of other quantitative reasoning problems (e.g., bin packing, knapsack, etc.). Real-world applications and extensions of number partitioning are numerous, including multi-processor scheduling [Sarkar, 1989; Pinedo, 2008], machine reassignment [Mehta *et al.*, 2012], and voting manipulation [Walsh, 2011]. Hence, efficient methods to determine optimal solutions are particularly relevant in the broader scope of combinatorial search.

Although number partitioning has served as a case study of optimal search techniques for nearly two decades, a renewed interest has emerged in recent years. In contrast to earlier approaches that recursively assign individual numbers to subsets [Korf, 1995], these new algorithms explore the space of potential pairwise *decompositions* in which entire subsets of numbers are generated in their entirety and partitioned independently [Korf, 2009]. Enhancements to this approach have considered more efficient means of iterating through viable subsets, as well as hybrid schemes that invoke different solving strategies depending on the values n and k [Korf, 2011]. A key insight shared by each of these previous algorithms is a *principle of optimality*, in which the optimality of the global solution is ensured by exploring only the space of *optimal* solutions to the elements in each decomposition.

While this divide-and-conquer strategy reduces the joint search space of subproblems, the cost of evaluating intermediate solutions remains high: there are exponentially-many decompositions, and the effort to resolve each one to optimality imposes a significant burden on the core optimization engine. Since successive decompositions are solved independently, a high degree of similar state is searched between repeated invocations of the subproblem solver. These deficiencies limit the efficacy of prior algorithms, especially for large k values that require many nested levels of bisection.

In this paper, we propose a new approach to multi-way number partitioning. Unlike previous algorithms, we consider the construction of potentially *suboptimal* decompositions, a task that is much easier to achieve and less time consuming. Under certain conditions, our approach may even neglect to produce solutions to subproblems if they exist. Yet, surprisingly, we are still able to provide the same properties as earlier works, including completeness, anytime behavior, and (most importantly) a guarantee of global optimality. To achieve this, we exploit a specific property of the objective function and develop a *principle of weakest-link optimality* that eliminates the need to consider purely optimal decompositions. We use this principle to strengthen the relationship between sequential decompositions by enforcing tighter bounds on partial assignments, and also to detect and remove dominated solutions from consideration. The efficiency of our algorithm is extremely competitive, outperforming the previous state-of-the-art by orders of magnitude and reducing average runtime on the largest benchmarks from several hours to less than a second. Empirical data also suggest that the performance of our algorithm is nearly independent of k.

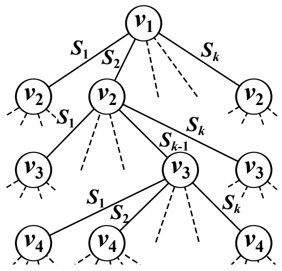

Figure 1: The space explored by the Complete Greedy Algorithm (CGA). Decision variables (v_1, v_2, ..., v_n) correspond to individual numbers. Branches (S_1, S_2, ..., S_k) correspond to partition assignments.

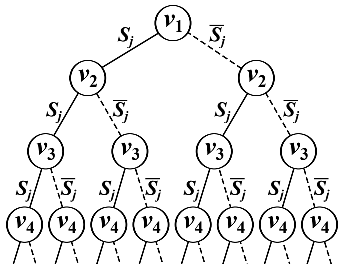

Figure 2: The space explored by Recursive Number Partitioning (RNP). Binary decision variables (v_1, v_2, ..., v_n) correspond to individual numbers. Leaf nodes correspond to decompositions for which optimal solutions must be generated.

2 Background

Exact approaches to number partitioning vary widely in how they model and explore the space of possible partitions. Here, we present an overview of prior approaches to this problem.

2.1 Complete Greedy Algorithm (CGA)

A simple greedy heuristic that produces high-quality (albeit suboptimal) solutions is to sort the numbers by size in decreasing order, and incrementally assign each one to the partition whose size is smallest [Graham, 1969]. This simple approach can be modified to compute optimal solutions by incorporating backtracking: each assignment of a number to a partition creates a different branch in search, and all possible combinations of assignments are attempted in order to produce the one with minimal cost (see Figure 1). By extending this depth-first framework with branch-and-bound pruning techniques and symmetry breaking, one arrives at the Complete Greedy Algorithm (CGA) [Korf, 1995; 1998]. CGA is thus a direct approach to combinatorial optimization, in which each number $\mathcal{S}_i$ corresponds to a decision variable v_i whose domain $D(v_i)$ is the set of possible subsets $\{S_1, S_2, ..., S_k\}$ that participate in the partition. All fully-instantiated complete assignments to these variables correspond to legal partitioning solutions.

2.2 Complete Karmarkar-Karp Algorithm (CKK)

An alternative greedy heuristic (named KK) constructs solutions in an entirely different way. Rather than committing each number to a specific partition, it instead considers potential pairings based on *set difference* [Karmarkar and Karp, 1982]. In each iteration of the algorithm, the two largest numbers are extracted from the set, and replaced by their absolute difference. In doing so, the numbers (or more generally, the subsets containing them) are ensured to be assigned to different partitions. The ultimate arrangement of these subsets is not finalized until the list is fully processed. A complete extension to this heuristic exists as well (the Complete Karmarkar-Karp algorithm, or CKK) [Korf, 1998]. The additional branches in this variation correspond to set unions, where numbers are joined rather than assigned to exclusive subsets. For $k = 2$, CKK performs dramatically faster than CGA (especially in the presence of perfect partitions) and can be further improved by pruning certain leaf nodes [Cerquides and Meseguer, 2012]. However, due to the high overhead and bookkeeping required, CKK is less effective for large k.

2.3 Recursive Number Partitioning (RNP)

Recall that in CGA, each number $\mathcal{S}_i$ is committed to a specific subset before the assignment of any subsequent numbers are processed. A radically different approach is taken by Recursive Number Partitioning (RNP) [Korf, 2009], in which a complete set of members for a range of partitions is constructed (and partitioned) before considering the members of subsequent partitions. If k is even, the set of partitions may be divided in half, and each individual number can be assigned to one half or the other. If k is odd, the partitions may be divided into S_1 and $[S_2, S_k]$. In order to explore the space of all possible arrangements, an inclusion-exclusion tree is searched (see Figure 2). Here, a binary decision variable v_i denotes the inclusion or exclusion of the number $\mathcal{S}_i$ from a set S_j. An internal node of this tree may be pruned if a partial subset cannot possibly extend to a better solution. The leaves of this tree correspond to coarse decompositions of numbers to subproblems, but not necessarily to assignments within each group. To construct solutions, optimal partitions are obtained for each subproblem, and combined if their concatenation results in an improved solution. Subproblems are solved in the same way; only in the base case (where $k = 2$) is traditional two-way partitioning employed. For certain values of n and k, RNP is several orders of magnitude faster than CGA.

2.4 Improved Recursive Number Partitioning

The basic framework of RNP has recently been enhanced in two fundamental ways to produce the Improved Recursive Number Partitioning (IRNP) algorithm [Korf, 2011]. First, the inclusion-exclusion tree is replaced by an extension of the SS algorithm [Schroeppel and Shamir, 1981] that more effectively searches the space of viable subset sums. This new approach, called ESS, divides the set of numbers in half and enumerates all possible subsets in each division. Although this requires exponential memory, it allows very fast exploration of satisficing sums simply by iterating through the two halves in tandem. Second, the solver makes calls to CGA (or CKK) for small n, since these problems tend to be more amenable to the original search space. Again, the efficacy of IRNP was shown to be orders of magnitude faster than previous works under certain conditions.

3 The Principle of Weakest-Link Optimality

Previous algorithms for multi-way partitioning have relied greatly on a so-called *principle of optimality*. To facilitate the following discussion, we define a $[k_1, k_2]$-decomposition of a partition $P = \langle S_1, S_2, ..., S_k \rangle$ to be a pair $\langle P_1, P_2 \rangle$ such that $P_1 = \langle S_1, S_2, ..., S_{k_1} \rangle$ and $P_2 = \langle S_{k_1+1}, S_{k_1+2}, ..., S_{k_1+k_2} \rangle$, where $k_1 + k_2 = k$. Its composition is written as $\langle P_1 \cdot P_2 \rangle$. We also define $\max(P_j)$ to be the maximum subset sum in P_j, and $opt(P_j)$ to be the maximum subset sum in any optimal repartitioning of P_j.

Theorem 1: Given any optimal k-way partition P and a $[k_1, k_2]$-decomposition $\langle P_1, P_2 \rangle$, any optimal k_1-way repartitioning of P_1 combined with any optimal k_2-way repartitioning of P_2 produces an optimal k-way partition P'. $\square$

Using this principle, it is possible to construct a globally optimal k-way partitioning P' by exploring the space of viable decompositions and computing optimal solutions to partitions smaller than k independently.

Corollary 1: There exists an optimal k-way partition $P' = \langle P_1 \cdot P_2 \rangle$, where $\langle P_1, P_2 \rangle$ is a $[k_1, k_2]$-decomposition of P', and both P_1 and P_2 are optimal. [Korf, 2009] $\square$

For any partition P' that satisfies this corollary, we denote it as being *semi-optimal* (in addition to being optimal in the traditional sense) since the elements of its decomposition are optimal as well. Unfortunately, the property of semi-optimality alone does not provide a sufficient condition for the global optimality of a given partition: that is, some semi-optimal solutions do not form a globally optimal composition.

Example: Consider a 3-way partitioning instance containing the numbers $\{1, 2, ..., 7\}$. Solution $S = \langle \{7, 3\}, \{6, 2, 1\}, \{5, 4\} \rangle$ is both optimal and semi-optimal with respect to a $[1, 2]$-decomposition, since partitions $\langle \{7, 3\} \rangle$ and $\langle \{6, 2, 1\}, \{5, 4\} \rangle$ are independently optimal with maximum subset sums of 10 and 9, respectively. However, while $S' = \langle \{7, 4\}, \{6, 3\}, \{5, 2, 1\} \rangle$ is also semi-optimal with respect to a $[1, 2]$-decomposition, its maximum subset sum is 11 and therefore is not globally optimal. $\square$

Because of this, all viable decompositions must be exhaustively considered to ensure the optimality of the composition. Furthermore, the requirement that these subproblems must be solved optimally themselves imposes a significant burden on even the most efficient implementation. In response, we propose a *principle of weakest-link optimality* that substantially reduces expectations on decomposition quality:

Theorem 2: Given any optimal k-way partition P and a $[k_1, k_2]$-decomposition $\langle P_1, P_2 \rangle$, any k_1-way repartitioning of P_1 where $\max(P_1) \leq opt(P)$ combined with any k_2-way repartitioning of P_2 where $\max(P_2) \leq opt(P)$ produces an optimal k-way partition P'. $\square$

The proof of this theorem is trivial. By definition, the cost of an optimal solution $opt(P)$ is simply equal to $\max(P)$. Furthermore, the maximum subset sum of a composition of elements must be at least as large as the largest maximum subset sum of each partition participating in its decomposition. A key difference between this theorem and the original principle of optimality is that one possible solution for P' is P itself; in other words, *any* optimal solution may be produced by the above construction.

Corollary 2: Every optimal k-way partition P is the combination $\langle P_1 \cdot P_2 \rangle$ of a $[k_1, k_2]$-decomposition where $\max(P_1) \leq opt(P)$ and $\max(P_2) \leq opt(P)$. $\square$

The principle of weakest-link optimality relaxes the requirement that the solutions to the decomposition are optimal; that is, each $\max(P_j)$ must be no greater than $opt(P)$, but it may potentially exceed $opt(P_j)$. This implies a fundamental consequence: semi-optimality is not a necessary condition for global optimality, hence optimal solutions may be composed from (potentially) *suboptimal* solutions to the subproblems of a decomposition.

Example (cont'd): Consider again the 3-way partitioning instance containing the numbers $\{1, 2, ..., 7\}$. Solution $S'' = \langle \{7, 3\}, \{6, 4\}, \{5, 2, 1\} \rangle$ is globally optimal, yet not semi-optimal: $\langle \{6, 4\}, \{5, 2, 1\} \rangle$ has a maximum subset sum of 10, whereas a lower sum of 9 is achievable. $\square$

The secret to this result lies in the sensitivity of the objective function: since only the *maximum* subset sum contributes to the quality of a solution (i.e., its "weakest link"), the relative balance between the remaining subsets is irrelevant. In some domains, objective functions that exhibit weakest-link optimality help to reduce the computational complexity required for exact optimization. For instance, in the case of constraint-based temporal reasoning, WLO-optimal solutions may be computed in polynomial time [Khatib *et al.*, 2001; 2003]. Within our context of number partitioning, the difficulty in exploiting the principle of weakest-link optimality when systematically generating solutions is that the value of $opt(P)$ is unknown in advance. Fortunately, it is possible to dynamically maintain upper and lower bounds on $opt(P)$ by integrating the intermediate results obtained between successive decompositions. This forms the basis of our approach, which is described in the following section.

4 A New Method

We now introduce our method for optimal k-way number partitioning. As in prior work, we define the cost of a partitioning $P = \langle S_1, S_2, ..., S_k \rangle$ to be the maximum subset sum, rather than the maximum difference between sets [Korf, 2010]; hence, the quantity:

$$\max_i \left(\sum_{s_{ij} \in S_i} s_{ij} \right)$$

is to be minimized. In addition to the parameter k and the original set $\mathcal{S}$ of numbers, our solver also takes as input a pair $\langle c_{min}, c_{max} \rangle$. Solutions whose costs are greater than c_{max} need not be considered, and any solution whose cost is c_{min} or less may be returned immediately. Optimal solutions to the original k-way partitioning may be obtained by passing $\langle \max(\mathcal{S}), \sum \mathcal{S} \rangle$. As will be shown later, more aggressive settings to these parameters can be used to communicate expectations between nested invocations of the algorithm.

4.1 Basic Framework

Similar to previous methods, we assemble partitions through recursive decomposition, where potential subsets are constructed by exploring an inclusion-exclusion binary tree.

However, in contrast to the state-of-the-art, our implementation does not consider balanced $[k/2, k/2]$ decompositions in the case of even k; this affords additional pruning capabilities (see Section 4.3) and also ensures consistent performance regardless of the parity of k. Each leaf within this tree corresponds to a complete assignment to a single partition subset $P_1 = \langle S_1 \rangle$, and all remaining numbers are assigned to subsequent subsets $P_2 = \langle S_2, S_3, ..., S_k \rangle$ through induction. Inclusion branches are always given precedence: whenever an element is assigned to the exclusion set $\overline{S_1}$, its inclusion in S_1 has already been considered (and/or pruned) earlier in search.

Our algorithm also enforces self-imposed upper and lower bounds on the sum of the first subset in order to prune partial assignments that cannot possibly lead to improved solutions. Whenever $\sum S_1 \geq c_{best}$, search may be aborted since the quality of the solution matches that of the best known partition. Furthermore, no better solution can possibly be found if $\sum \overline{S_1} \geq c_{best} * (k-1)$, and so in these cases search may be terminated as well. These bounds provide a powerful means to prune partial assignments, and they are employed by previous solvers.

4.2 Bounding Suboptimal Decompositions

Once a candidate subset is established for $P_1 = \langle S_1 \rangle$, a recursive call is invoked to resolve the subproblem $P_2 = \langle S_2, S_3, ..., S_k \rangle$. According to the classical principle of optimality, there exists some optimal solution to one of these decompositions that produces an optimal solution to the original problem.

However, previous examples have demonstrated that optimal solutions may exist that do not require strict optimality within the decomposition. Suboptimal solutions are generally easier to obtain [Thayer and Ruml, 2011], although they must still meet a minimum bar of quality in order to extend to globally optimal results. Theorem 2 tells us that for any $[k_1, k_2]$-decomposition to an optimal partition P, it must be the case that $\max(P_1) \leq opt(P)$. Although $opt(P)$ is unknown, we can rest assured that if a solution to P_2 happens to be found such that $\max(P_2) \leq \max(P_1)$, it must also be the case that $\max(P_2) \leq opt(P)$ (satisfying the property of global optimality). Furthermore, if some solution $S_{best} = \langle P'_1 \cdot P'_2 \rangle$ has already been found at the highest level with cost c_{best}, any solution (optimal or otherwise) to P_2 where $\max(P_2) \geq c_{best}$ is clearly fruitless; any such decomposition would be incapable of improving upon the global objective function.

In order to limit the search performed by child solvers for P_2 and its descendants, we pass the parameters $\langle c_{min}, c_{max} \rangle = \langle \max(P_1), c_{best} \rangle$. Partial assignments are abandoned whenever $\sum S_i \geq c_{max}$ or $\sum \overline{S_i} \geq c_{max} * (k-i)$. Furthermore, a complete solution S will be returned whenever $\sum S_i \leq c_{min}$ for all $i \leq k$. Note that these are distinct from the self-imposed upper and lower bounds, since they are determined by criteria computed by parent solvers. The reasoning behind these extensions should be intuitive, as they extend directly from the weakest-link nature of the partitioning objective function: a decomposition cannot be redeemed if its cost exceeds that of solutions encountered at higher levels, yet it receives no extra credit for obtaining sums below its neighboring partitions.

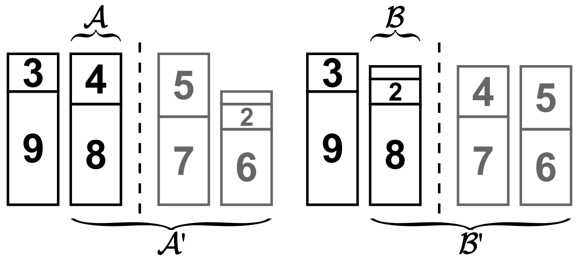

Figure 3: Extending partial assignment $\mathcal{A}$ to $\mathcal{A}'$ supports a globally optimal solution. Partial assignment $\mathcal{B}'$ supports a semi-optimal solution, but cannot improve the global result.

Example: Consider the 4-way partitioning instance containing the numbers $\{1, 2, ..., 9\}$. A decomposition that assigns $S_1 = \{9, 3\}$ will impose the directive $c_{min} = 12$ when partitioning the remaining numbers among the subsets $\{S_2, S_3, S_4\}$. For this subproblem, the partial assignment $\mathcal{A} = \langle S_2 \rangle = \langle \{8, 4\} \rangle$ extends to a complete assignment $\mathcal{A}'$ which satisfies this criterion (see Figure 3). An algorithm that requires strict semi-optimal decompositions must continue to exhaustively enumerate additional assignments, such as $\mathcal{B} = \langle S_2 \rangle = \langle \{8, 2, 1\} \rangle$. These attempts may succeed in improving the local cost of the subproblem (e.g., assignment $\mathcal{B}'$ achieves a locally optimal cost of 11), but they cannot improve the cost of the global solution. Hence, our algorithm prunes $\mathcal{B}$ entirely and returns $\mathcal{A}'$ to the topmost level. □

4.3 Dominance Detection and Elimination

The previous section employed tighter bounding criteria to prune partial assignments. In this section, we introduce a different form of pruning, in which *dominated* solutions (i.e., solutions that are cost-preserving transmutations of earlier assignments) are detected and eliminated from search. Set dominance techniques have proven particularly effective in related studies of bin packing [Korf, 2003] and bin completion [Fukunaga and Korf, 2005], in which bins are assigned a fixed (rather than variable) capacity.

Consider a partial assignment $S = \langle S_1, \overline{S_1} \rangle$ and two subsequent extensions for the next number $\mathcal{S}_i$ to be assigned:

- its inclusion branch $S^+ = \langle S_1^+ = S_1 \cup \{\mathcal{S}_i\}, \overline{S_1^+} \rangle$
- its exclusion branch $S^- = \langle S_1^-, \overline{S_1^-} = \overline{S_1} \cup \{\mathcal{S}_i\} \rangle$

We assume that the branch for S^+ has been taken, and that the alternate branch for S^- is being considered.

Theorem 3: If $\sum S_1^+ < c_{max}$, then in any complete solution S^c that extends S^- where $\max(S^c) < c_{max}$, it must be the case that $\sum S_1^c > \sum S_1^+$. □

The above theorem makes a rather bold claim: after considering the inclusion of a number $\mathcal{S}_i$ in a set whose sum is less than the upper bound, any subsequent extension to the set must strictly *exceed* that sum to obtain an improved solution.

Proof: Suppose by contradiction that there exists a solution S^c for which $\max(S^c) < c_{max}$ and $\sum S_1^c \leq \sum S_1^+$. The set of remaining numbers added to S_1 in S^c are $S_1^r = S_1^c - S_1$. Recall that S^c extends S^-, therefore the number $\mathcal{S}_i$ must be assigned to some subset S_j^c where $j > 1$. We can construct

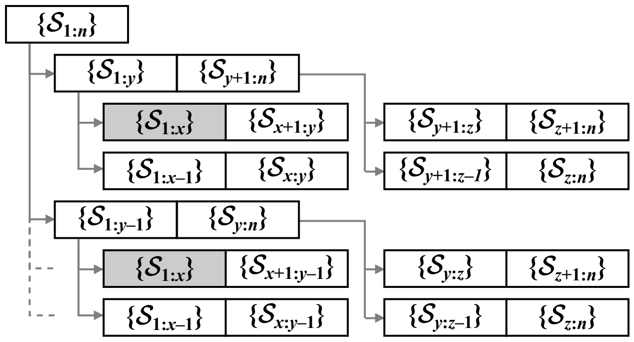
$\{\mathcal{S}_{1:n}\}$
$\{\mathcal{S}_{1:y}\}$ $\{\mathcal{S}_{y+1:n}\}$
$\{\mathcal{S}_{1:x}\}$ $\{\mathcal{S}_{x+1:y}\}$
$\{\mathcal{S}_{1:x-1}\}$ $\{\mathcal{S}_{x:y}\}$
$\{\mathcal{S}_{y+1:z}\}$ $\{\mathcal{S}_{z+1:n}\}$
$\{\mathcal{S}_{y+1:z-1}\}$ $\{\mathcal{S}_{z:n}\}$
$\{\mathcal{S}_{1:y-1}\}$ $\{\mathcal{S}_{y:n}\}$
$\{\mathcal{S}_{1:x}\}$ $\{\mathcal{S}_{x+1:y-1}\}$
$\{\mathcal{S}_{1:x-1}\}$ $\{\mathcal{S}_{x:y-1}\}$
$\{\mathcal{S}_{y:z}\}$ $\{\mathcal{S}_{z+1:n}\}$
$\{\mathcal{S}_{y:z-1}\}$ $\{\mathcal{S}_{z:n}\}$

Figure 4: Redundant search performed by RNP and IRNP. For any constants $\{x, y, z\}$, the subproblem $\{\mathcal{S}_1, ..., \mathcal{S}_x\}$ (along with three other subproblems illustrated) will be solved multiple times. This pattern repeats at all depths, and does not depend on subset contiguity.

a new solution S' from S^c by exchanging contents between two of the subsets:

$$S'_1 = S^c_1 \cup \{\mathcal{S}_i\} - S^r_1$$
$$S'_j = S^c_j \cup S^r_1 - \{\mathcal{S}_i\}$$

In the above construction, element $\mathcal{S}_i$ is essentially swapped with all the numbers that were appended to S^-_1 in forming S^c_1. Since $\sum S^c_1 \leq \sum S^+_1$, it must be the case that $\sum(S^c_1 - S_1) \leq \sum(S^+_1 - S_1)$, which is equivalent to $\sum S^r_1 \leq \mathcal{S}_i$. This implies that $\sum S'_j \leq \sum S^c_j$, i.e., the sum of subset j does not increase in our new solution. In contrast, the sum of the first subset may potentially increase: $\sum S'_1 \geq \sum S^c_1$. However, note that S'_1 is simply equal to S^+_1, and therefore we are guaranteed that $\sum S'_1 < c_{max}$. No other subsets have been modified, so we conclude that $\max(S') < c_{max}$. However, since S' is an extension of S^+, it has already been discovered in the inclusion branch, implying that $\max(S') \geq c_{max}$. These results are inconsistent, and therefore S^c cannot exist. $\square$

Through similar reasoning, we can also prune leaf nodes (that is, any assignment where all numbers have been assigned to either S_1 or $\overline{S_1}$) in which there exists a number $\mathcal{S}_i \in \overline{S_1}$ such that $\mathcal{S}_i + \sum S_1 \leq c_{min}$. Although dominance detection is sound for any variable ordering strategy, it clearly benefits from a descending sort of the numbers in $\mathcal{S}$, since individual numbers smaller than $\mathcal{S}_i$ will be pruned unless they are combined with others.

4.4 Pruning Symmetric Solutions

For every solution $P = \langle S_1, S_2, ..., S_k \rangle$, there are $k!$ equivalent symmetric solutions that can be obtained by permuting the relative ordering of subsets. Many of these isomorphic assignments can be eliminated by artificially imposing a total ordering over subsets. Recent works have achieved this by requiring subset sums to be non-decreasing, such that $\sum S_j \leq \sum S_{j'}$ for any $j < j'$. Unfortunately, such a policy defers the construction of the most influential subset (i.e., the one with maximal sum) and severely limits opportunities for pruning. We instead adopt a variation of a technique originally developed for CGA, in which a number is never assigned to more than one empty subset. This is accomplished by pruning the exclusion branch for the largest number in any subproblem solver. If the numbers are stored in decreasing order, the number $\mathcal{S}_1$ will be forced into subset S_1, $\mathcal{S}_2$ will be assigned to subset S_2 (assuming it is excluded from S_1), and so on.

$$\min \quad c_{max} \tag{1}$$
$$\sum_{i \in [1,n]} \mathcal{S}_i \times v_{ij} \leq c_{max} \qquad \forall j \in [1, k] \tag{2}$$
$$\sum_{j \in [1,k]} v_{ij} = 1 \qquad \forall i \in [1, n] \tag{3}$$
$$0 \leq v_{ij} \leq 1 \qquad \forall i \in [1, n], j \in [1, k] \tag{4}$$
$$v_{ij} \text{ is integer} \qquad \forall i \in [1, n], j \in [1, k] \tag{5}$$

Figure 5: A mixed-integer program for number partitioning

4.5 Comparison to IRNP

Our overall approach relies on fast computation of bounded suboptimal solutions to subproblems, eliminates partial assignments that are provably dominated by those considered earlier in search, and employs stronger techniques for symmetry breaking; these are the primary advantages over IRNP, the best multi-way number partitioner in the literature [Korf, 2011]. A secondary advantage is that our memory requirements are linear rather than exponential, since ESS is not used to generate subset sums. Our tighter bounding criteria could potentially be extended to the subset enumeration techniques within IRNP, although it considers candidates out-of-order and therefore cannot easily adopt our implementation of dominance detection. For small to medium values of n, the top-level of IRNP simply calls the old CGA solver, which is better equipped to optimize a large number of small partitions. In contrast, we do not use CGA for these settings since our algorithm is substantially faster.

5 Experimental Results

To determine the efficacy of our approach, we compare our algorithm to the latest results on optimal multi-way number partitioning as reported in [Korf, 2011]. Our solver is implemented in less than one hundred lines of C++ code and executed on a 3.47GHz Intel Xeon Processor. We consider the largest suite of benchmarks in the literature where k ranges from 7 to 10, and n is varied between 20 and 40. In each benchmark, n random numbers are generated uniformly in the range $[0, 2^{31} - 1]$. For every setting of k and n, runtimes are averaged over a set of one hundred instances.

Table 1 provides the results of these experiments. Our solver dramatically outperforms IRNP on all test cases; furthermore, the runtime gap between the two algorithms consistently widens with increased k. At the highest setting of $k = 10$, our approach demonstrates an improvement of up to three or even four orders of magnitude. For the largest set of problems solved by both algorithms, IRNP requires nearly *two hours* on average, whereas the runtime of our solver remains well under *one second*.

	Seven Way			Eight Way			Nine Way			Ten Way		
n	IRNP	Ours	Ratio	IRNP	Ours	Ratio	IRNP	Ours	Ratio	IRNP	Ours	Ratio
20	.005	.000	> 1×	.001	.000	> 1×	.000	.000	> 1×	.000	.000	> 1×
21	.012	.000	> 1×	.007	.000	> 7×	.001	.000	> 1×	.000	.000	> 1×
22	.038	.000	> 1×	.020	.000	> 20×	.005	.000	> 5×	.001	.000	> 1×
23	.062	.001	62×	.068	.000	> 68×	.076	.000	> 76×	.011	.000	> 11×
24	.131	.001	131×	.339	.000	> 339×	.206	.000	> 206×	.041	.000	> 41×
25	.235	.002	117×	.937	.001	937×	.438	.000	> 438×	.061	.000	> 61×
26	.435	.002	217×	:02	.002	1000×	:01	.001	1000×	:01	.001	1000×
27	.791	.003	263×	:04	.003	1333×	:05	.002	2500×	:02	.001	2000×
28	:01	.005	200×	:09	.004	2250×	:11	.005	2200×	:08	.003	2666×
29	:03	.011	272×	:21	.009	2333×	:46	.006	7666×	:28	.006	4666×
30	:04	.012	333×	:37	.012	3083×	:59	.014	4214×	1:10	.012	5833×
31	:06	.024	250×	:48	.022	2181×	1:43	.016	6437×	11:33	.062	11177×
32	:09	.036	250×	1:16	.039	1948×	7:01	.029	14517×	15:17	.022	41681×
33	:16	.053	301×	2:07	.057	2228×	7:13	.051	8490×	40:07	.051	47196×
34	:36	.097	371×	3:13	.095	2031×	10:05	.096	6302×	1:07:59	.099	41202×
35	:42	.158	265×	5:15	.135	2333×	15:01	.184	4896×	1:53:57	.157	43547×
36	1:14	.244	303×	8:24	.227	2220×	25:29	.243	6292×		.232	
37	1:56	.412	281×	13:04	.353	2220×	38:18	.367	6261×		.531	
38	3:24	.663	307×	24:04	.661	2184×	1:16:30	.583	7873×		.747	
39	5:11	.998	311×	42:03	:01	2360×	2:03:13	.927	7975×		:01	
40	8:59	:02	299×	1:20:43	:02	2936×		:02			:02	

Table 1: Average Time to Optimally Partition 31-bit Integers Seven, Eight, Nine, and Ten Ways

Perhaps most surprising is that the performance of our algorithm scales almost independently of k. Recall that while k increases the set of available partitions for a number, it also reduces both the effective capacity for each partition in any optimal solution and (consequently) the number of viable subset combinations. Since only the arrangement of numbers within this area is ultimately affected, problems with higher k are not necessarily harder. Because our tighter pruning criteria propagate upper and lower bounds between nested decompositions, our algorithm tends to perform well under these conditions. We performed additional experiments with higher values of n, and found that problems with up to 56 numbers can be solved in roughly 100 minutes or less, regardless of k. This appears to be the largest problem size we can solve within a two-hour time limit, and is well beyond the capabilities of published works.

Several additional techniques were considered to further improve the performance of our approach, but most were found to be subsumed by the aforementioned innovations. One of these was *memoization*, a simple and well-known method that caches results of subproblems to avoid needless recomputation. As shown in Figure 4, RNP and IRNP both appear to be highly susceptible to redundant search, since successive decompositions tend to share considerable overlap (forcing solvers at lower levels to optimize identical subsets multiple times). To remedy this, solutions to intermediate problems can be stored and subsequently retrieved by using subset contents as the index into a hash table or prefix tree. However, we observed that when our bounding criteria and dominance techniques were both enabled, this enhancement rarely removed duplicate nodes, and often performed slower due to the bookkeeping of intermediate results and memory overhead.

We also explored solving strategies that rely on mixed-integer programming (MIP), a powerful and widely popular optimization paradigm that (to our knowledge) has never before been considered in prior works on number partitioning. The MIP encoding in Figure 5 creates a binary indicator variable v_{ij} for every number-partition pair, constraining each number to be assigned to exactly one partition. We used CPLEX to solve this formulation, but found that its performance came nowhere near the speed of the previous state-of-the-art. An additional complication is that CPLEX cannot ensure optimality for our benchmarks due to precision limitations: even in the 64-bit version, values and bounds on integer variables (including the objective c_{max}) must not exceed $2^{31} - 1$, and continuous variables are modeled by floating-point types which are subject to roundoff error. Nevertheless, improved MIP encodings may be viable for test cases requiring lower precision, and is a topic worthy of continued study.

6 Conclusions

We have proposed a new approach to multi-way number partitioning, one that produces optimal results despite relying on the solutions of potentially *suboptimal* decompositions. To achieve this, we have exploited a specific property of the objective function to develop a *principle of weakest-link optimality*. We have used this principle to strengthen the relationship between sequential decompositions by enforcing tighter bounds on partial assignments, and also to detect and remove dominated solutions from consideration. The efficiency of our algorithm is extremely competitive, outperforming the previous state-of-the-art by orders of magnitude and reducing average runtime on the largest benchmarks from several hours to less than a second.

References

[Cerquides and Meseguer, 2012] Jesús Cerquides and Pedro Meseguer. Speeding up 2-way number partitioning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 223–227, 2012.

[Fukunaga and Korf, 2005] Alex S. Fukunaga and Richard E. Korf. Bin-completion algorithms for multicontainer packing and covering problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 117–124, 2005.

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

[Graham, 1969] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[Karmarkar and Karp, 1982] Narendra Karmarkar and Richard M. Karp. The differencing method of set partitioning. In *Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley*, 1982.

[Khatib *et al.*, 2001] Lina Khatib, Paul Morris, Robert Morris, and Francesca Rossi. Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 322–327, 2001.

[Khatib *et al.*, 2003] Lina Khatib, Paul Morris, Robert Morris, and K. Brent Venable. Tractable Pareto optimal optimization of temporal preferences. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 1289–1294, 2003.

[Korf, 1995] Richard E. Korf. From approximate to optimal solutions: A case study of number partitioning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 266–272, 1995.

[Korf, 1998] Richard E. Korf. A complete anytime algorithm for number partitioning. *Artif. Intell.*, 106(2):181–203, 1998.

[Korf, 2003] Richard E. Korf. An improved algorithm for optimal bin packing. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 1252–1258, 2003.

[Korf, 2009] Richard E. Korf. Multi-way number partitioning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 538–543, 2009.

[Korf, 2010] Richard E. Korf. Objective functions for multiway number partitioning. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SOCS 2010)*, 2010.

[Korf, 2011] Richard E. Korf. A hybrid recursive multi-way number partitioning algorithm. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 591–596, 2011.

[Mehta *et al.*, 2012] Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Comparing solution methods for the machine reassignment problem. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012)*, pages 782–797, 2012.

[Pinedo, 2008] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, third edition, 2008.

[Sarkar, 1989] Vivek Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA, 1989.

[Schroeppel and Shamir, 1981] Richard Schroeppel and Adi Shamir. A $T = O(2^{\mathrm{n}/2})$, $S = O(2^{\mathrm{n}/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

[Thayer and Ruml, 2011] Jordan Tyler Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 674–679, 2011.

[Walsh, 2011] Toby Walsh. Where are the hard manipulation problems? *J. Artif. Intell. Res. (JAIR)*, 42:1–29, 2011.