A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem

Robin Lamarche-Perrin, Yves Demazeau, Jean-Marc Vincent

Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem

(revised version: October 2014)

by

Robin Lamarche-Perrin, Yves Demazeau, and Jean-Marc Vincent

Preprint no.: 105 2014

A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem

Preprint* of the Max-Planck-Institute for Mathematics in the Sciences
October 6th, 2014

Robin Lamarche-Perrin
MPI for Mathematics in the Sciences, Leipzig, Germany
Robin.Lamarche-Perrin@mis.mpg.de

Yves Demazeau
CNRS, Laboratoire d'Informatique de Grenoble, France
Yves.Demazeau@imag.fr

Jean-Marc Vincent
Univ. Grenoble Alpes, Laboratoire d'Informatique de Grenoble, France
Jean-Marc.Vincent@imag.fr

Abstract

Given a set of individuals, a collection of admissible subsets, and a cost associated to each of these subsets, the *Set Partitioning Problem* (SPP) consists in selecting admissible subsets to build a partition of the individuals that minimizes the total cost. This combinatorial optimization problem has been used to model dozens of problems arising in specific domains of Artificial Intelligence and Operational Research, such as coalition structures generation, community detection, multilevel data analysis, workload balancing, image processing, and database optimization. All these applications are actually interested in *special versions* of the SPP: the admissible subsets are assumed to satisfy global algebraic constraints derived from topological or semantic properties of the individuals. For example, admissible subsets might form a hierarchy when modeling nested structures, they might be intervals in the case of ordered individuals, or rectangular tiles in the case of bidimensional arrays. Such constraints structure the search space and – if strong enough – they allow to design tractable algorithms for the corresponding optimization problems. However, there is a major lack of unity regarding the identification, the formalization, and the resolution of these strongly-related combinatorial problems. To fill the gap, this article proposes a generic framework to design specialized dynamic-programming algorithms that fit with the algebraic structures of any special versions of the SPP. We show how to apply this

*An early and shorter version of this paper has been published in the *Proceedings of the 2014 IEEE International Conference on Tools with Artificial Intelligence* (ICTAI'14) [27] and the full document has been submitted in October 2014 to the *Artificial Intelligence* journal.

framework to two well-known cases, namely the *Hierarchical SPP* and the *Ordered SPP*, thus opening a unified approach to solve versions that might arise in the future.

Keywords: Combinatorial Optimization, Set Partitioning Problem, Structural and Semantical Constraints, Algebraic Structure of Partition Lattices, Dynamic Programming, Specialized Optimization Algorithms, Artificial Intelligence, Operational Research.

1 Introduction

The *Set Partitioning Problem* (SPP) is a deeply-studied combinatorial optimization problem that naturally arises as soon as one wants to organize a set of objets into covering and pairwise disjoint subsets such that an additive objective is optimized. Within this setting, the *population* designates the finite set of individuals that needs to be partitioned. The *set of admissible parts* is the collection of nonempty subsets of the population that can be used to do so. It fully determines the *set of admissible partitions*, that is the set of all partitions of the population that can be built from these admissible parts, thus constituting the search space of the SPP. Finally, the *cost function*, associating a real value to each admissible part, is the objective that one wants to minimize. It is additively extended to the search space by associating to each partition the sum of the costs of its parts. For further details, the SPP will be fully formalized in Subsection 3.1.

This optimization problem has been used to model a colossal amount of problems in the neighboring fields of Artificial Intelligence and Operational Research. In each of these applications, the admissibility constraints are used to express the structural or semantic properties of the population that needs to be partitioned. This is for example the case in algorithmic game theory for *coalition structure generation* [4, 5, 44, 45, 49]. A population of self-motivated agents must be partitioned into coalitions that optimally exploit the interagent synergies to achieve a given task. Because of compatibility constraints and the agent spatiotemporal locations, all coalitions are not necessarily feasible in practice. The set of admissible parts allows to express which coalitions can be actually considered to organize the agents depending on the system's structures and functions. Moreover, because of communication costs and (dys)synergies, some coalitions might be more effective than the others to perform the task. The cost function then expresses the social welfare of admissible coalitions. For another example, in data analysis, *multilevel aggregation* consists in partitioning a set of data points into homogeneous classes. In order to be meaningfully interpreted by the analyst, the classes often need to preserve the system's semantic or topological structure, expressed by the set of

admissible parts [28, 30, 38]. In this context, the cost function is usually defined according to a compression rate that should be maximized, leading to numerous applications such as *time series analysis* [26, 28, 40], *database optimization* and *image processing* [35,38], *multilevel community detection* [43], *performance analysis* of distributed systems [20, 30], *resource and task allocations* for multiagent systems [50]. Subsection 2.1 will present in further details the whole range of application of the SPP.

The SPP is NP-complete in the *general* case [12]. This still holds when all admissible parts have the same cost [46], when they do not contain more than three individuals [46], when every individual is contained in exactly two admissible parts [31], or when admissible partitions consist in exactly two admissible parts [2]. Hence, even with these very restrictive additional constraints, one cannot hope for a general-purpose algorithm that can efficiently solve every instance of the SPP. Among the strategies that have been proposed to tackle this computational challenge (see Subsection 2.2), this paper is interested in approaches that consider global constraints regarding the set of admissible parts in order to reduce and structure the search space. Such constraints, arising from global (semantic, structural, functional, or topological) properties of the modeled population, thus define tractable *special versions* of the SPP. For example, in data aggregation, the dataset's topological properties may allow to define a class of search spaces that are meaningful for the domain expert: *e.g.*, assuming that admissible parts are *intervals* in the case of time series analysis [26, 28, 40], that they form a *hierarchy* in the case of spatial analysis of nested structures [28, 30, 43], or that they are *rectangular tiles* in the case of image processing, where the two-dimensional structure of the dataset implies very strong constraints for compression [38]. In this paper, we thus assume that the expert provides such global constraints to reduce the SPP computational complexity and provide tractable optimization algorithms.

Although many such special versions of the SPP have been addressed in the past (see Section 3), there is a major lack of unity regarding the identification, the formalization, and the resolution of these however strongly-related problems. Many of the articles herein referenced deal with special versions of the SPP with domain specific objectives and do not benefit from the research that have been extensively done by the combinatorial optimization community. Consequently, some results have been proven several times by independent work, such as the *Ordered Set Partitioning Problem* (see Subsection 3.4) that has been solved in quadratic time at least five times in 30 years [2,12,26,46,53]. Rothkopf *et al.* have addressed several versions of the SPP in an integrating work [46], leading to deep results regarding their tractability in a unified *applicative* context, but without proposing any unified *algorithmic* framework to solve them. Some other work characterized the tractability of special versions of the SPP by identifying general algebraic prop-

erties of the corresponding sets of admissible parts [36, 37, 39]. However, such models are often too general, and thus too weak in practice (meaning that more specific combinatorial algorithms might perform better on a special version than these general models) or, in some cases, they do not provide any practical algorithm [37]. In order to fill the gap, this paper proposes to look more carefully at some particular classes of admissibility constraints and at the combinatorial algorithms that optimally exploit their algebraic structure. Then, we propose a generic algorithmic framework that can be applied to any special version of the SPP in order to design a specialized optimization algorithm.

This paper is organized as follows. Section 2 presents in further details the broad range of applications and methods of the SPP. Section 3 identifies several special versions that have been addressed in the past, along with the specialized algorithm that have been proposed. Section 4 presents our generic algorithmic framework that can be used to design such specialized algorithms. It relies on a proper understanding of the search space's algebraic properties and on dynamic programming to efficiently exploit them. Section 5 applies this framework to two special versions of the SPP, namely the *Hierarchical Set Partitioning Problem* (HSPP) and the *Ordered Set Partitioning Problem* (OSPP). We show that the computational complexity of the resulting optimization algorithms meets the one of past algorithms dedicated to the same problems [2, 12, 28, 43, 46, 53], thus opening a unified approach to solve new versions of the SPP that might arise in the future. Section 6 discusses the limitation and possible generalizations of our framework when the admissibility constraints and the cost function are defined "at the partition level". Lastly, Section 7 summarizes the results and gives some research perspectives.

2 Related Work

This section introduces to related work regarding the SPP in two steps. Subsection 2.1 presents some major applications in Operational Research and in Artificial Intelligence. Subsection 2.2 then identifies three main categories of approaches that aim at solving the SPP. It also justifies the positioning of this paper regarding the third approach, detailed in Section 3. Table 2 provides a double entry summary of the articles making the connexion between the presented applications and approaches.

2.1 Applying the SPP

2.1.1 Relation to Other Combinatorial Optimization Problems

The prolific applicability of the SPP is partly explained by its closeness to the extensively-studied *Set Packing* (SP) and *Set Covering* (SC) *problems* [6], respectively corresponding to the relaxation of the "covering" and the "pairwise disjoint" constraints regarding the concept of admissible partitioning. The SPP also generalizes numerous combinatorial optimization problems, in particular in *computational graph theory* where individuals are vertices of a graph and admissible parts are defined according to a specific subgraph structure: *e.g.*, *graph coloring*, *graph partitioning*, *domatic partitioning*, *weighted multiway cut*, *minimum set cover*, *partition into Hamiltonian subgraphs, forests, or perfect matching* [9, 13, 36], thus leading to a tremendous amount of other possible applications. However, since this paper focuses on applications where cost functions and sets of admissible parts are provided by expert knowledge regarding the analyzed population and its semantical or structural properties, this subsection only presents in further details applications in specialized fields of Artificial Intelligence and Operational Research.

2.1.2 The *airline crew scheduling problem* and other operational problems

Historically, the best-known application of the SPP in Operational Research is the *airline crew scheduling problem* [3, 6]: given a set of flight legs (individuals), a set of sequences of legs that are feasible by airline crews according to work policies (admissible parts), and a cost associated to each feasible sequence, the airline company would like to find a collection of feasible sequences minimizing the total cost such that each flight leg is covered by exactly one crew. This setting is easily generalizable to a broader class of *transportation, delivery, routing, and location problems* [6, 10, 24, 25], and to other well-known operational problems such as the *circuit partitioning problem*, where a set of electronic components should be divided into clusters such that the number of intercluster connections is minimized [1, 13], or the *political districting problem*, where regions must be divided into voting districts such that every citizen is assigned to exactly one district [6, 24]. Many other examples of such applications can be found in more exhaustive surveys: *e.g.*, *stock cutting*, *line and capacity balancing*, *facility location*, *capital investment*, *switching current design*, *marketing* [2, 6].

2.1.3 The *winner determination problem* in combinatorial auctions

Given a set of assets (individuals) and a set of bids associating a price to groups of asset (admissible parts), the auctioneer wants to find an allocation of the assets

to the bidders that maximizes her revenue [31, 37, 39, 46, 48, 50]. Due to the fact that several bidders may give a price to the same collection of assets, the *winner determination problem* is strictly-speaking more general than the SPP. However, Rothkopf *et al.* [46] have shown that, when considering the classical *OR bidding language*[1] to express the bidder objectives, the *winner determination problem* is equivalent to the *Set Packing problem* [46]. Moreover, since the bids are always positive, and by interpreting the absence of bid on single assets as null prices [48], there is always an optimal packing that is also an optimal partition. Hence, in this context, the *winner determination problem* is equivalent to the SPP.

2.1.4 The *coalition structure generation problem* in algorithmic game theory

A population of self-motivated agents (individuals) must collaborate with one another to perform a given task. Coalition structure generation consists in partitioning the agents into feasible teams (admissible parts) so as to achieve better result by maximizing the social welfare (expressed by the cost function) [4, 5, 44, 45, 49]. This field itself leads to many applications in *e-commerce* (buyers form coalitions to purchase a product in bulk and take advantage of price discounts), *e-business* (groups form to satisfy particular market niches), *distributed sensor network* (sensors work together to track targets), *distributed vehicle routing* (coalition of delivery companies to reduce transportation costs), and *information gathering* (servers form coalitions to answer queries) [45].

2.1.5 The *clustering problem* for multilevel data analysis

The SPP can also be seen as a general formulation of the classical *clustering problem*: data points have to be partitioned into classes such that the intra-class similarity and/or the inter-class dissimilarity are maximized [9, 11, 42]. Finding the optimal partition thus arises as soon as one wants to organize, to classify, or to abstract data. In this context, the SPP also relates to *data aggregation problems*, where data points are partitioned into homogeneous classes preserving the system's structure and optimizing a given compression rate, leading to applications in *time series analysis* [26,28,40], *spacial analysis* [20,28,30], *multilevel community detection* [43], *database optimization*, and *image processing* [35,38].

[1] In this context, the bidders are willing to pay for any combination of pairwise disjoint collections a price equal to the sum of the bid prices they expressed for these collections [31].

2.2 Solving the SPP

The SPP is usually tackled by one of the following strategies or by mixing several of them in a portfolio approach (for more detailed surveys regarding optimization algorithms for the SPP over the last forty years, see [6, 8, 10, 14, 24, 25]).

2.2.1 Exploiting the Algebraic Structure of the General SPP

The set of parts and the set of partitions have several useful algebraic properties when one tries to directly tackle the general problem by going through the whole search space of admissible partitions. Strategies formalizing and exploiting such properties to efficiently run through the search space are usually based on *integer programming* [6,21,24,25], *dynamic programming* [2,26,44,46,48,54], *exhaustive enumeration* [48], *implicit enumeration* [6, 18, 25, 34], and/or *automatic reformulation* of the linear description [25, 47, 52]. Again, since the SPP is NP-complete, one should not expect any worst-case polynomial algorithm to emerge from such strategies, unless P = NP.

Heuristics limiting the search space in some way have also been proposed to find suboptimal solutions in reasonable time, including *genetic algorithms* [14], *dual ascent* [10], *simulated annealing*, and *neural networks* [24], and other *metaheuristic algorithms* [8]. However, such approaches do not provide any worst-case guarantee regarding the closeness to optimality [45]. On the contrary, *approximation algorithms* provide provable solution quality and run-time bounds [4, 5, 9, 13, 31, 38], but are still limited by severe inapproximability results [48] and the absence of any *polynomial-time approximation scheme* [31].

2.2.2 Exploiting Properties of the Cost Function

Much work has focused on special versions of the SPP by making additional assumptions regarding the cost function. For example, the SPP has been proved to be polynomially solvable when costs are defined by some aggregative measures (*e.g.*, max-sum, min-sum, sum-max, sum-avg) applied to the attributes of the individuals [2, 35, 38]. More constrained settings have been studied, such as aggregative measures applied to the edges of a weighted graph expressing the synergies, dissimilarities, communication costs, or interests in grouping couples of individuals [4, 5, 11, 36, 42]. However, in each of these cases, the considered cost function requires a dedicated treatment that can hardly be generalized to a broader context.

Other work has hence focused on more general properties of the cost function such as *concavity* [2, 12], *symmetry* [13], *submodularity* [2, 13, 31, 36, 37], *superadditivity* [38, 49], *subadditivity* [49], and *additivity with convex discounts* [37]. These approaches all assume that the costs are somehow monotonously defined

with respect to the set inclusion. For example, *superadditivity* in coalition games implies that the synergies between agents cannot decrease when new agents join a coalition. Hence, applying additional cuts will never degrade the quality of a partitioning, and one usually search for a partition of a given fixed size that optimizes the superadditive cost function [38]. *Subadditivity* implies, on the contrary, that the agents are usually best of by operating alone. Although considerable results have been achieved for such settings, Sandholm *et al.* [49] argue that, when some cost penalizes the coalition formation process itself (because of communication or anti-trust penalties), many applications of the *coalition generation problem* are neither superadditive nor subadditive. This is also the case in multilevel data analysis, where the information-theoretic measures are usually non-monotonous regarding the set inclusion, meaning that data points might be relatively homogeneous at some level, but heterogeneous at lower or higher levels [20, 28, 30, 40, 43].

2.2.3 Exploiting Structures of the Admissible Parts

This paper focuses on a third category of strategies exploiting global constraints on the set of admissible parts to define easier versions of the SPP. Indeed, by assuming that admissible parts belong to a restraint and structured portion of the population power set, one can reduce the search space and thus provide tractable optimization algorithms. In this case, one should guarantee that the constraints do not exclude solutions that would be optimal otherwise. For example, in the *winner determination problem*, if the assets are known to be more valuable in given combinations, the auctioneer may anticipate the bids of greatest economic significance and only allow such valuable combinations. For example, in the presence of a topological structure, groups of neighboring assets may be more valuable than random groups [31, 46]. The auctioneer thus assumes that no bidder will prefer forbidden combinations during the auction process. Reducing the search space thus consists in "introducing patterns that have a meaningful auction interpretation" [37]. However, constraints might also arise from semantics considerations when some subsets are not meaningful for the partition purposes. In data aggregation, for example, the partitioning should be consistent with the dataset's structural and topological properties so that the compressed data is usable by the domain expert [20, 28, 30, 40]. In particularly, in image processing, the two-dimensional structure implies very strong constraints for compression [38].

Some work has also been dedicated to the characterization of tractability by studying general algebraic properties of the sets of admissible parts. For example, it has been shown that, if the coefficient matrix of the corresponding integer programming problem is *totally unimodular*, then the linear relaxation of the SPP has an integral optimal solution [36, 37, 39]. In this case, a linear programming solver

provides an optimal solution in polynomial time (see for example the column generator algorithm proposed in [36]). Another characterization of tractability is based on the concept of *maximal cliques of perfect graphs*: if the intersection graph of admissible parts is perfect and if all maximal cliques in this graph are induced by the population individuals, then a maximum weighted stable set can be computed in polynomial time [37]. However, such models of tractability might be too *general*, and thus too weak in practice. Even if they allow to show that some special versions of the SPP can be solved in polynomial type, they do not imply that linear programming solvers will provide the best possible algorithm. Indeed, in the presence of *more specific* combinatorial structures, dedicated algorithms might perform better by exploiting the structure more precisely than the general models [37]. Moreover, it has been argued that in some cases such models do not provide any practical algorithm [37].

In order to overcome these limitations, this paper proposes to look more carefully at some particular classes of admissibility constraints to derive specialized combinatorial algorithms that optimally exploit such structures. In Section 4, we thus propose a generic framework based on dynamic programming to provide such *optimal* algorithms. Note that other mixed strategies have been proposed, such as assuming global admissibility constraints to design specialized *approximation* algorithms (2.2.1) [48], or by making assumptions regarding the set of admissible parts *and* the cost function (2.2.2) to design optimal algorithms [36]. On the contrary, the framework proposed in this paper both allows to design optimal algorithms and to stay fully general regarding the optimized cost function.

3 The General SPP and Some Special Versions

This section formalizes the general SPP (3.1) and presents some special versions that have been addressed in previous work (3.2, 3.3, 3.4, 3.5, and 3.6). For each version, we identify applications in Artificial Intelligence and Operational Research by indicating possible semantical and structural interpretations of global admissibility constraints. The third part of Table 2 summarizes the links between the applications and these special versions. We also identify algorithms that have been proposed in previous work to optimally solve these versions (see Table 1).

3.1 The General Set Partitioning Problem (SPP)

Preliminary Notations. This paper uses a consistent system of letter cases to properly formalize the SPP and its search space:

- *individuals* are designated by lowercase letters: x, y, z;

- sets of individuals and *parts* by uppercase letters: Ω, X, Y, Z;
- sets of parts and *partitions* by calligraphic letters: $\mathcal{P}$, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$;
- sets of partitions by Gothic letters. For example: $\mathfrak{P}$, $\mathfrak{R}$, and $\mathfrak{C}$.

A *population* $\Omega = \{x_1, \dots, x_n\}$ is a finite set of individuals and a *part* is a nonempty subset $X \subset \Omega$. A *set of admissible parts* $\mathcal{P} = \{X_1, \dots, X_m\}$ is a subset of the power set 2^Ω. The *size* of a population or a part, resp. marked $|\Omega|$ and $|X|$, is the number of individuals it contains. An *admissible partition* $\mathcal{X} = \{X_1, \dots, X_k\} \subset \mathcal{P}$ is a set of covering and pairwise disjoint admissible parts: $X_i \in \mathcal{P}$, $\bigcup_i X_i = \Omega$ and $X_i \cap X_j = \varnothing$. The *set of admissible partitions* $\mathfrak{P}$ is the set of all partitions that can be generated from the set of admissible parts: $\mathfrak{P} = \{\mathcal{X} \subset \mathcal{P} \setminus \mathcal{X} \text{ is a partition of } \Omega\}$. We assume that $\mathcal{P}$ is such that $\mathfrak{P} \neq \varnothing$, so that the SPP is indeed an *optimization* problem, and not an *existence* problem. This is for example the case when the population is admissible, so that the *maximal partition* is admissible: $\Omega \in \mathcal{P} \Rightarrow \{\Omega\} \in \mathfrak{P}$, or when all singletons are admissibles, so that the *minimal partition* is admissible: $\forall x \in \Omega, \{x\} \in \mathcal{P} \Rightarrow \{\{x\}\}_{x \in \Omega} \in \mathfrak{P}$. The *size* of a partition, marked $|\mathcal{X}|$, is the number of parts it contains. A *cost function* c is an application that associates to each admissible part $X \in \mathcal{P}$ a real value $c(X) \in \mathbb{R}$. We also define the additive extension of the cost function c on the set of admissible partitions: $\forall \mathcal{X} \in \mathfrak{P},\ c(\mathcal{X}) = \sum_{X \in \mathcal{X}} c(X)$. Lastly, we mark $\mathfrak{P}^* \subset \mathfrak{P}$ the set of admissible partitions that minimizes c.

The Set Partitioning Problem. Given a population Ω, a set of admissible parts $\mathcal{P}$, and a cost function c, the *weighted Set Partition Problem* (SPP) *with an additive objective* consists in finding a subset $\mathcal{X}^* \subset \mathcal{P}$ that partitions Ω and minimizes c:

$$\mathcal{X}^* \in \operatorname*{arg\,min}_{\mathcal{X} \in \mathfrak{P}} c(\mathcal{X}). \tag{1}$$

Tractability of the SPP and its Special Versions. With no further assumption regarding the cost function and the set of admissible parts, the SPP is NP-complet [12]. However, the rest of this section presents special versions of the problem where global admissibility constraints allow to define tractable algorithms. In the following, we thus indicate the worst-case time complexity of optimization algorithms proposed by previous work according to the size $|\Omega| = n$ of the population. The algorithmic complexity is thus given relatively to the *problem size* and we consider a special version *tractable* if there exists an algorithm that optimally

solves any instance of this special version by requiring a number operations that is polynomially bounded in the size of the population.

Note however that the *input size* of the problem is actually the number of admissible parts $|\mathcal{P}| = m$. Indeed, the encoding length of any instance of the SPP is in general the number of cost values that have to be specified. In that case, the SPP can easily become intractable because the feeding of the input is itself intractable [46]. However, in many work, the costs of admissible parts are actually derived from attributes of the individuals [3, 7, 20, 26, 28, 30, 35, 38, 40, 43] or from relations between individuals [4, 5, 7, 11, 42]. This is the case for example in multilevel data analysis when the cost of admissible parts are computed in linear time from the individual attributes according to some information-theoretic measure [20, 28, 30]. In such cases, the size of the input thus linearly depends on the number of individual, and not on the number of admissible parts. Hence, most of the special versions presented hereafter are *linearly* solvable with respect to the *input size* $|\mathcal{P}| = m$, but *polynomially* solvable with respect to the *problem size* $|\Omega| = n$ (see Table 1 for a detailed list of the input sizes of the following problems).

3.2 The Complete Set Partitioning Problem (CSPP)

The *Complete Set Partitioning Problem* arises when all subsets of the population are admissible, *i.e.*, when the set of admissible parts is the population's power set: $\mathcal{P} = 2^{\Omega}$.

Applications. The CSPP has been extensively used to model *coalition structure generation problems* assuming that every possible group of agents is an adequate candidate to constitute a coalition [44,45,49]. It has also been applied to *corporate tax structuring* to find an optimal aggregation of corporate subsidiaries to pay state unemployment compensation tax [54].

Combinatorics. Sandholm *et al.* [48] have shown that the number of admissible partitions grows considerably faster than the number of admissible parts: $|\mathcal{P}| = \Theta(2^n)$ and $|\mathfrak{P}| = \omega(n^{n/2})$.

Algorithmic Results. The CSPP is NP-complete [48,49]. However, exponential algorithms have been provided to solve small instances of the CSPP: a $O(2^{n^{n/2}})$ enumeration approach [32], a $O(3^n)$ dynamic programming algorithm [31, 54], and a $O(n^n)$ anytime algorithm that quickly generates a suboptimal solution with

bound guarantees, then slowly improves this solution by establishing better bounds [44,45,49]. Note that, since any instance of the SPP can be modeled as an instance of the CSPP with infinite cost on non-admissible parts, these algorithms can also be exploited to solve instances of the general problem. However, as it is shown in what follows, when assuming stronger admissibility constraints, more efficient algorithms can be designed.

3.3 The Hierarchical Set Partitioning Problem (HSPP)

The *Hierarchical Set Partitioning Problem* arises when the set of admissible parts is a *hierarchy*, *i.e.*, when every two admissible parts are either disjoint or one is included in the other: $\forall (X_1, X_2) \in \mathcal{P}^2,\ X_1 \cap X_2 = \varnothing$ or $X_1 \subset X_2$ or $X_1 \supset X_2$. If one also assumes that the population and the singletons are admissible: $\Omega \in \mathcal{P}$ and $\forall x \in \Omega,\ \{x\} \in \mathcal{P}$, then the hierarchy can be described as a *rooted tree*[2] where the *leaves* represent the singletons, the *nodes* represent the admissible parts the *root* represents the whole population, and the *tree-order* is the subset relation (see Fig. 1).

Applications. The HSPP has been mainly applied in data aggregation to model systems with multilevel nested structures. This include *community representation of networks* (individuals are nodes of a graph representing a social structure and admissible parts are highly-connected groups of nodes resulting from a hierarchical community detection algorithm) [43], *aggregation of geographical data* for geographical analysis (individuals are territorial units and admissible parts are defined according to nested geographical partitions of the world into regions, countries, subcontinents, continents, *etc.*) [28], and *analysis of distributed systems* for performance analysis (individuals are the computational resources of a distributed system and admissible parts are defined according to the system's hierarchical communication network: processes, machines, clusters, sites, *etc.*) [20, 30].

Combinatorics. For a population of size n, the minimal number of admissible parts is $n+1$ (singletons and population) and the maximum number is reached for a hierarchy corresponding to a complete binary tree. Thus, $|\mathcal{P}| = O(n)$. The number of admissible partitions however depends on the number of levels and branches

[2]Note that, if the population or the singletons are not admissible, but if the union of all admissible parts is: $\bigcup_{X \in \mathcal{P}} X \in \mathcal{P}$, then the hierarchy can still be represented as a rooted tree. In any case, the hierarchy can be represented as a *forest of rooted trees* and the algorithms solving the HSPP can easily be generalized by a sequential execution to each tree of the forest.

in the hierarchy. Since an admissible partition of $X \in \mathcal{P}$ is either the maximal partition $\{X\}$ or the union of admissible partitions of subparts of X, this number can be recursively defined as $|\mathfrak{P}(X)| = 1 + \prod_{Y \sqsubset X} |\mathfrak{P}(Y)|$, where $\sqsubset (X)$ is the set of children of X in the tree representing the hierarchy. For a complete binary tree, at each level, the number of admissible partitions is squared. Assuming that $n = 2^k$, where k is the depth of the tree, we thus have $|\mathfrak{P}| = U_k = 1 + (U_{k-1})^2$ with $U_0 = 1$. The sequence $(U_k)_{k \in \mathbb{N}}$ is asymptotically bounded by an exponential function α^n, with $\alpha \approx 1.226$ [15]. Similar results are found for complete ternary trees (with $\alpha \approx 1.084$ [33]), complete quaternary trees, and so on. Henceforth, for any bounded number of children per node, the number of admissible partitions exponentially grows with the population size.

Results. All algorithms that have been proposed to solve the HSPP consist in a $O(n)$ depth-first search of the tree representing the hierarchy [28, 30, 43]. A $O(n|\mathcal{P}|)$ dynamic algorithm has also been proposed [46].

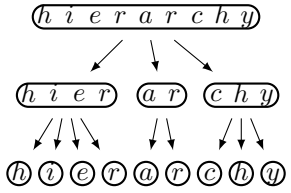

Figure 1: A 3-levels hierarchy defined on a population of size 9

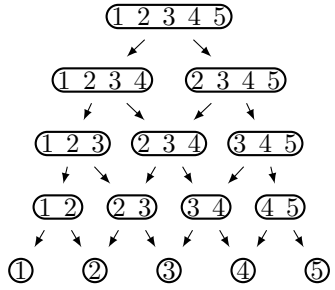

Figure 2: The "pyramid of intervals" of an ordered population of size 5

3.4 The Ordered Set Partitioning Problem (OSPP)

The *Ordered Set Partitioning Problem* arises when a total order $<$ is defined on Ω and when the admissible parts are the intervals induced by this order: $\mathcal{P} = \{\{x_i < \ldots < x_j\} \subset \Omega \setminus i \leqslant j\}$. This set can be represented as a "pyramid of intervals" (see Fig. 2) and the resulting admissible partitions are sometimes called *consecutive partitions* [2].

Applications. The OSPP very naturally applies to model any population that has a temporal component (*e.g.*, sets of dates, events, or time periods are naturally or-

dered by the "arrow of time"). For example, the OSPP has been addressed for the aggregation of time series [26, 28, 40] (individuals are time periods and admissible parts are consecutive sequences of such periods) and for the allocation of consecutive processing time to different tasks [37]. This setting might also receive a unidimensional-space interpretation, such as the North-South geographical ordering of the cities on the East Coast or the mineral rights on tracks forming a single swath of offshore state waters [46]. It has also been applied to *inventory control* and *production planing* [2, 12, 53].

Combinatorics. The number of intervals of an ordered population of size n is $\frac{n(n+1)}{2}$. An admissible partition of size k consists in cutting the population in $k-1$ places. Hence, the number of partitions of size k is $\binom{n-1}{k-1}$ and the total number of partitions is: $|\mathfrak{P}| = \sum_{k=0}^{n-1} \binom{n-1}{k} = 2^{n-1}$.

Results. Chakravarty *et al.* [12] have shown that, when the optimal partition is a sequence of intervals, solving the SPP is equivalent to solving the *shortest path problem*, resulting in a $O(n^2)$ optimization algorithm. A $O(n^2)$ dynamic programming algorithm has also been proposed in independent work [2, 26, 46, 53].

Extensions. Some other order-related structures have been addressed, such as *monotone partitions* (when the size of intervals increases with the order) [2], *extremal partitions* (the last individual of each interval can be associated to the next interval without losing optimality) [2], and *cyclic orders* defining intervals on a circle and leading to a $O(n^3)$ dynamic algorithm [46].

3.5 The Array Partitioning Problem (APP)

The *Array Partitioning Problem* consists in partitioning a two-dimensional array into rectangular tiles. It naturally arises when one considers the Cartesian product $\Omega_1 \times \Omega_2$ of two ordered populations. In that sense, we write APP = OSPP $\times$ OSPP. The set of admissible parts is then $\mathcal{P} = \{X \subset \Omega_1 \times \Omega_2 \setminus \exists X_1 \in \mathcal{P}_1, \exists X_2 \in \mathcal{P}_2, X = X_1 \times X_2\}$, where $\mathcal{P}_1$ and $\mathcal{P}_2$ are respectively the sets of intervals of Ω_1 and Ω_2 (see the OSPP above).

Applications. As for the OSPP, the APP may be used to model spatial structures such as the rectangular partitioning of geographic locations on a two-dimensional grid [7, 46]. This includes the distribution of individual units of a manufacturing

plant among supervisors, the balanced subdivision of a rectangular mining area among mining companies [7], the clustering of points with fairly uniform color or similar frequencies in image processing, computer graphics, and video compression [35, 38], the building of histograms based on rectangular regions to approximate multi-dimensional data distributions in database systems [38]. Moreover, the APP has been used to model *load balancing problems* in parallel computation when the computational space corresponds to a matrix. In this case, the tiles are rectangular in order to respect the computation space topology [38] and to reduce the communication costs between subproblems that is proportional to the number of adjacent pairs in the grid [35].

Combinatorics. Given the Cartesian product $\Omega_1 \times \Omega_2$ of a population of size n_1 and a population of size n_2, the number of rectangular tiles is $|\mathcal{P}| = |\mathcal{P}_1| \times |\mathcal{P}_2| = \Theta(n_1^2 n_2^2)$. To get a lower bound estimate of $|\mathfrak{P}|$, we recall that the number of interval partitions of Ω_1 is 2^{n_1-1} (see combinatorics of OSPP above). Thus, by only considering "stacks" of n_2 partitions of Ω_1, the total number of admissible partitions of the APP is necessarily greater than $2^{(n_1-1)n_2}$.

Results. Rothkopf *et al.* [46] have shown that the APP is NP-complet even if one only considers singletons and 2×2 rectangles as admissible parts. To the best of our knowledge, no optimal algorithm has yet been proposed.

Extensions. The APP is manageable if one only considers rows, columns, and singletons, for example to represent assets that have two different properties of interest for a collector, as the year and denomination of a coin [46]. Other multidimensional versions of the SPP has been addressed, such as the partition of d-dimensional hypercubes [46] or the Cartesian product of a hierarchy and a total order (HSPP $\times$ OSPP), leading to a $O(n_1(n_2)^3)$ dynamic algorithm [20], where n_1 and n_2 are respectively the sizes of the hierarchical and the ordered populations.

3.6 Bounds on the Size of Admissible Parts

Some versions of the SPP assume that the size of admissible parts is bounded: $\mathcal{P} = \{X \subset \Omega \setminus |X| \leqslant k\}$ or $\mathcal{P} = \{X \subset \Omega \setminus |X| = 1 \text{ or } |X| > n/k\}$ for a given $k \in \mathbb{N}$.

Applications. Such assumptions are very generic and might apply to any problem bringing in only small groups (or only large groups) of individuals. It has been proposed for example to model slot auctions of airline take-off and landing [37].

Results. Rothkopf *et al.* [46] have shown that, when admissible parts are limited to size $k = 3$, the SPP is still NP-complete, when limited to size $k = 2$, solving the SPP is equivalent to solving the *maximum-weight matching problem*, leading to a $O(n^3)$ optimization algorithm, and when only large parts and singletons are admissible, the SPP can be solved in $O(n|\mathcal{P}|^k)$ time.

		$\lvert\mathcal{P}\rvert$	$\lvert\mathfrak{P}\rvert$	Best Known Algorithm
CSPP (3.2)		$\Theta(2^n)$	$\omega(n^{n/2})$	NP-complet
HSPP (3.3)	Binary Tree	$\Theta(n)$	$\Theta(\alpha^n)$	$O(n)$ [28, 43]
OSPP (3.4)	Total Order	$\Theta(n^2)$	$\Theta(2^n)$	$O(n^2)$ [26, 46]
	Cyclic Order	$\Theta(n^2)$	$\Theta(2^n)$	$O(n^3)$ [46]
APP (3.5)	OSPP $\times$ OSPP	$\Theta(n_1^2 n_2^2)$	$\Omega(2^{n_1 n_2})$	NP-complet
	HSPP $\times$ OSPP	$\Theta(n_1 n_2^2)$	$\Omega(2^{n_1 n_2})$	$O(n_1 n_2^3)$ [20]
	Rows & Columns	$\Theta(n_1 n_2)$	$\Theta(2^{n_1} + 2^{n_2})$	$O(n_1 n_2)$ [46]

Table 1: Combinatorics and optimal algorithms of special versions of the SPP

	Operational Research		**Artificial Intelligence**	
	Scheduling, Delivery, Transportation (2.1.2)	*Winner Determination Problem* (2.1.3)	*Coalition Structure Generation* (2.1.4)	*Multilevel Data Clustering* (2.1.5)
Solving the General Problem (2.2.1)				
Optimal Algorithms	[2] [6] [24] [25]	[46] [48]		
Heuristics/Approximation	[24] [10]	[31] [48]	[45] [4] [5]	[38]
Exploiting Algebraic Properties of the Cost Function (2.2.2)				
Specific Measures	[2]		[4] [5]	[38] [35] [11] [42]
General Properties	[12] [2]	[31] [37]	[49]	[38]
Exploiting Algebraic Properties of the Admissible Parts (2.2.3)				
CSPP (3.2)	[54]	[31] [48]	[49] [44] [45]	
HSPP (3.3)		[46]		[30] [28] [43]
OSPP (3.4)	[12] [2] [53]	[46] [37]		[28] [26] [40]
APP (3.5)	[7]	[46]		[38] [35] [20]
SPP with Bounds (3.6)		[46] [37]		

Table 2: Summary of previous work regarding application and resolution of the *Set Partitioning Problem*

4 A Generic Algorithmic Framework to Solve Special Versions of the SPP

If dynamic programming has been used to solve the general SPP, leading to a $\Omega(2^n)$ and $O(3^n)$ optimization algorithm [46,48], we have shown in previous section that more efficient algorithms are possible when dealing with special versions of the problem. Dynamic programming has thus been used on many occasions to solve such tractable versions (see for example [2, 26, 44, 46, 54]). However, these results are very independent from each other and no unified framework have been proposed for their generalization. To fill the gap, the framework proposed in this paper aims at being *generic* – in the sense that it can be applied to any special version of the SPP to derive a specialized algorithm – but not at being *general* – in the sense that it would not be efficient to deal with the general unstructured problem. Hence, this framework should be considered as an abstract tool to build optimization algorithms for tractable versions of the SPP. Section 5 gives two examples of such *specialized implementations* (for the HSPP and the OSPP).

The key principle regarding the application of dynamic programming to the SPP is the following: one needs to compute the optimal partition of an admissible part only once to evaluate all the partitions that are coarser than this part. This principle has been used to straightforwardly solve the general SPP [46, 48]. In this section, in order to also exploit the particular structure of special versions, we provide a better understanding of the search space that allows to identify and suppress numerous redundant computations when applying this key principle. The search space is first broken down into smaller covering subspaces. Then, thanks to a *principle of optimality* that fits with the algebraic structure of the partition set, these subproblems are recursively solved. Locally-optimal solutions are then compared to globally solve the initial problem.

The rest of this section is organized as follows. We first formalize the algebraic structure of the search space and we provide a corresponding *principle of optimality* (4.1). Then, we propose a decomposition of the search space based on this principle (4.2) and we derive a recursive algorithm (4.3). Finally, we propose two improvements based on *memoization* and *non-redundant decomposition* to fully exploit the algebraic structures of special versions (4.4). The final algorithm is given at the end of this section.

4.1 Algebraic Structure and Principle of Optimality

Rothkopf *et al.* have argued that the computational complexity of the SPP does not actually depend on the size of the search space, but rather on its structure [46]. Indeed, as shown in Table 1, restricting the number of admissible parts $|\mathcal{P}|$ is not

sufficient to restrict the size of the search space $|\mathfrak{P}|$. For example, in the case of the OSPP, we have $|\mathcal{P}| = \Theta(n^2)$ and $|\mathfrak{P}| = \Theta(2^n)$. Since the number of admissible partitions grows exponentially with the population size in most of the special versions reported in Section 3, the introduction of strong constraints is often not sufficient to make a brute-force search algorithm tractable. Hence, in this subsection, we carefully examine the algebraic structure of the search space and we propose a *principle of optimality* that exploits this structure to evaluate partitions in a computationally-efficient fashion.

4.1.1 The Algebraic Structure of the Search Space

The set of admissible partitions $\mathfrak{P}$ is structured by an essential algebraic relation, usually referred to as the *refinement relation* $\subset$ [17]. A partition $\mathcal{X}$ *refines* a partition $\mathcal{Y}$ ($\mathcal{X} \subset \mathcal{Y}$), if and only if each part in $\mathcal{X}$ is a subset of a part in $\mathcal{Y}$:

$$\mathcal{X} \subset \mathcal{Y} \quad \Leftrightarrow \quad \forall X \in \mathcal{X}, \quad \exists Y \in \mathcal{Y}, \quad X \subset Y$$

As this binary relation is reflexive, antisymmetric, and transitive, it defines a partial order on the partition set $\mathfrak{P}$ that consequently forms a poset and can be represented as a Hasse diagram [17].

The *covering relation* $\sqsubset$ is the transitive reduction of the refinement relation, that is the binary relation which holds between immediate "neighbors" regarding $\subset$. Hence, a partition $\mathcal{X}$ is *covered* by a partition $\mathcal{Y}$ ($\mathcal{X} \sqsubset \mathcal{Y}$), if and only if $\mathcal{X} \neq \mathcal{Y}$, $\mathcal{X}$ refines $\mathcal{Y}$, and there is no other partition "between" them:

$$\mathcal{X} \sqsubset \mathcal{Y} \quad \Leftrightarrow \quad \mathcal{X} \subsetneq \mathcal{Y} \quad \text{and} \quad \nexists \mathcal{Z} \in \mathfrak{P}, \ \mathcal{X} \subsetneq \mathcal{Z} \subsetneq \mathcal{Y}$$

As it is shown in the rest of this section, these two relations give essential algebraic tools to search for optimal partitions. For a given admissible partition $\mathcal{X} \in \mathfrak{P}$, we define $\mathfrak{R}(\mathcal{X})$ as the *set of admissible partitions refining* $\mathcal{X}$, $\mathfrak{C}(\mathcal{X})$ as the *set of admissible partitions covered by* $\mathcal{X}$ and, respectively, $\mathfrak{R}^*(\mathcal{X})$ and $\mathfrak{C}^*(\mathcal{X})$ as the sets of optimal partitions among $\mathfrak{R}(\mathcal{X})$ and $\mathfrak{C}(\mathcal{X})$. Note that if the *minimal partition* $\{\{x\}\}_{x\in\Omega}$ is admissible, it refines all admissible partitions: $\mathcal{X} \in \mathfrak{P}, \ \{\{x\}\}_{x\in\Omega} \in \mathfrak{R}(\mathcal{X})$, and if the *maximal partition* $\{\Omega\}$ is admissible, it is refined by all admissible partitions: $\mathfrak{R}(\{\Omega\}) = \mathfrak{P}$.

4.1.2 A Principle of Optimality for the SPP

In dynamic programming, finding a principle of optimality consists in showing that the search space has an optimal substructure: the solution to the optimization problem can be obtained by recursively combining locally-optimal solutions to several

subproblems. Intuitively, in the case of the SPP, one can rely on the fact that *the union of optimal partitions on subsets of the population is an interesting candidate to form an optimal partition of the whole population*. Hence, by appropriately decomposing the population, one might provide a computationally efficient procedure to build such an optimal solution.

Theorem 1. *Let Ω be a population and c be an additive cost function defining partition optimality. For any partition $\mathcal{Y}$ of Ω, the union of locally-optimal partitions of the parts of $\mathcal{Y}$ is optimal among the refinements of $\mathcal{Y}$:*

$$\forall Y \in \mathcal{Y},\ \mathcal{Y}_Y^* \in \mathfrak{P}^*(Y) \quad \Rightarrow \quad \left(\bigcup_{Y \in \mathcal{Y}} \mathcal{Y}_Y^* \right) \in \mathfrak{R}^*(\mathcal{Y}) \tag{2}$$

Proof. *The proof of this theorem is given in annexe.*

4.2 Branching the Search Space

Given a admissible part $X \in \mathcal{P}$ for which one wants to compute a locally-optimal admissible partition $\mathcal{X}^* \in \mathfrak{P}^*(X)$, a *branching* consists in building subspaces $\mathfrak{P}_1, \ldots, \mathfrak{P}_k$ that cover the search space: $\mathfrak{P}_1 \cup \ldots \cup \mathfrak{P}_k = \mathfrak{P}(X)$. Then, if one finds locally-optimal partitions $\mathcal{X}_1^* \in \mathfrak{P}_1^*, \ldots, \mathcal{X}_k^* \in \mathfrak{P}_k^*$ for each of these subspaces, one can easily solve the optimization problem the following way:

$$\underset{\mathcal{X} \in \{\mathcal{X}_1^*, \ldots, \mathcal{X}_k^*\}}{\arg\min} \quad c(\mathcal{X}) \quad \subset \quad \mathfrak{P}^*(X) \tag{3}$$

For that purpose, the covering relation indicates "atomic disaggregations" of a given part. For example, in the case of the OSPP, it consists in dividing the part into *two* intervals (see Fig. 3). The covering relation can thus be used to branch the search space. First, assuming that the maximal partition $\{X\}$ is admissible[3], we know that all admissible partitions of X refine the maximal partition $\{X\}$: $\mathfrak{P}(X) = \mathfrak{R}(\{X\})$. Second, for any partition $\mathcal{X} \in \mathfrak{P}(X)$, a refining partition of $\mathcal{X}$ is either *the partition $\mathcal{X}$ itself*, or *a partition that refines a partition covered by $\mathcal{X}$*. Hence, the search space can be branched the following way:

$$\mathfrak{P}(X) = \{\{X\}\} \cup \left(\bigcup_{\mathcal{Y} \in \mathfrak{C}(\{X\})} \mathfrak{R}(\mathcal{Y}) \right) \tag{4}$$

[3] If not, the following approach can easily be generalized by sequentially applying the algorithm to all maximal partitions, *i.e.* maximal elements in the poset of partitions induced by the refinement relation.

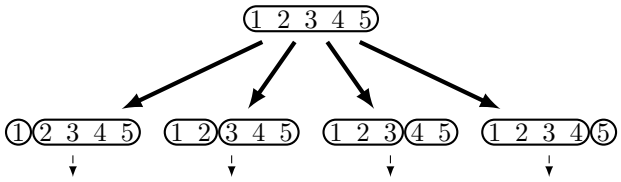

Figure 3: Decomposition of the search space according to the covering relation in the case of an ordered population of 5 individuals

4.3 A Recursive Algorithm

The computation of an optimal partition thus consists in computing *locally-optimal partitions refining the partitions covered by the maximal partition.* Thanks to the principle of optimality, such a computation can be recursively performed by recursively applying the algorithm on admissible parts (see Eq. 2 and Fig. 4). Hence, the three branching and recursion equations 2, 3, and 4 allow to define a divide and conquer algorithm that computes a locally-optimal partition $\mathcal{X}^* \in \mathfrak{P}^*(X)$ for any $X \in \mathcal{P}$ according to the following recursive formula:

$$\mathcal{X} \in \{\{X\}\} \cup \left(\underset{\mathcal{Y} \in \mathfrak{C}(\{X\})}{\bigcup} \left\{ \underset{Y \in \mathcal{Y}}{\bigcup} \mathcal{Y}_Y^* \right\} \right) \qquad \underset{c(\mathcal{X}) \subset \mathfrak{P}^*(X)}{\arg\min} \tag{5}$$

where $\mathcal{Y}_Y^*$ designates a partition in $\mathfrak{P}^*(Y)$. Here are the steps of the resulting algorithm:

(step 1) Compute the set $\mathfrak{C}(\{X\})$ of admissible partitions covered by the maximal partition $\{X\}$.

(step 2) For each partition $\mathcal{Y} \in \mathfrak{C}(\{X\})$, do the following:

- (step 2.a) for each part $Y \in \mathcal{Y}$, recursively compute a locally-optimal admissible partition $\mathcal{Y}_Y^* \in \mathfrak{P}^*(Y)$;
- (step 2.b) compute the union $\mathcal{Y}^* = \bigcup_{Y \in \mathcal{Y}} \mathcal{Y}_Y^*$ of these partitions. The principle of optimality ensures that $\mathcal{Y}^* \in \mathfrak{R}^*(\mathcal{Y})$.

(step 3) Return a partition that minimizes c among $\{X\}$ and the $\mathcal{Y}^* \in \mathfrak{R}^*(\mathcal{Y})$ computed for each $\mathcal{Y} \in \mathfrak{C}(\{X\})$.

Fig. 5 gives an example of execution of this algorithm in the case of the OSPP with an ordered population of size 4. The starting point is the maximal partition at the top (1 2 3 4). The plain numbered arrows represent the sequence of branchings executed by the algorithm (step 2): for example branching [1] evaluates the covering partition (1 2 3)(4). The dashed arrows represent the recursive calls on an admissible part (step 2.a): for example (1 2 3)(4) corresponds to the execution of the algorithm on the first part of the partition (1 2 3)(4). Crosses and stars are explained in the next subsection.

4.4 Dynamic Programming Improvements

This first algorithm is not computationally-optimal in terms of execution steps. In the rest of this subsection, we thus propose two improvements to reduce its time complexity. The fist one, simply consisting in the *recording of intermediary results* (see stars), is actually a key principle of dynamic programming [46, 48]. The second one, consisting in *avoiding redundant evaluations* in the execution tree (see crosses), is an entirely new feature derived from the algebraic analysis of the search space.

4.4.1 Recording Intermediary Results

According to the dynamic programming paradigm, recursive algorithms can be easily improved by recording the results of time-consuming recursive calls. For each part on which the algorithm is once applied, by keeping trace of the resulting locally-optimal partition, one can immediately return this result when posterior calls occur on the same part. This way, the algorithm is applied only once to each admissible part $X \in \mathcal{P}$. For example, in Fig. 5, the algorithm is initially applied *twice* on parts (1 2), (2 3), and (3 4). Thanks to this *memoization* procedure, one can avoid the second calls (see stars on dashed lines).

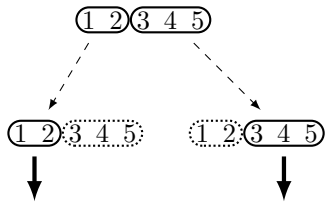

Figure 4: Recursive application of the algorithm

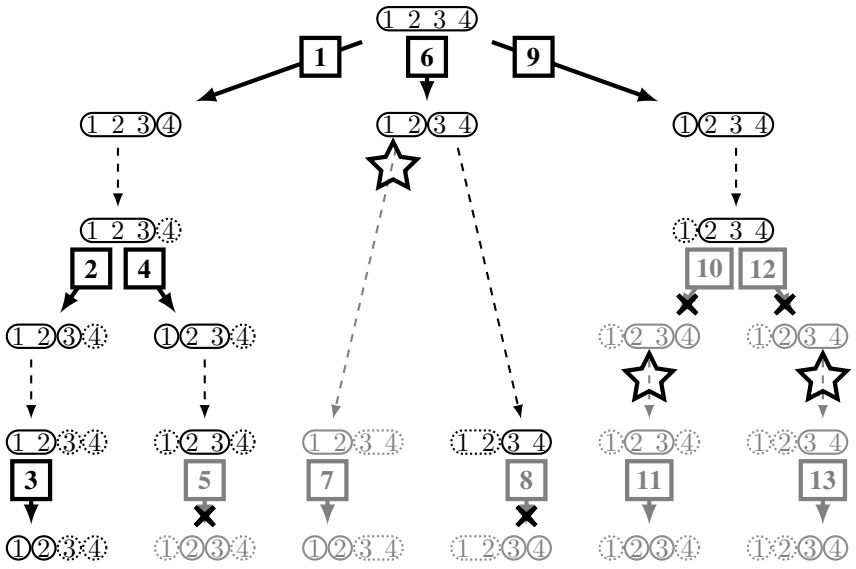

Figure 5: Execution trace of the recursive algorithm (Subsection 4.3) for an ordered population of size 4. Plain numbered arrows represent branchings (step 2). Dashed arrows represent recursive calls (step 2.a). Crosses and stars give the cuttings of branches that may improve the algorithm (Subsection 4.4).

4.4.2 Avoiding Redundant Evaluations

The branching of the search space proposed in Eq. 4 is redundant, *i.e.*, subspaces are not disjoints. For example, in Fig. 5, branches **2** and **3** allow the evaluation of partitions ①②③④ and ①②③④, and branches **4** and **5** the evaluation of partitions ①②③④ and ①②③④. Hence, ①②③④ is evaluated *twice* and **5** is useless. In order to avoid such redundant branches, one can keep trace of the covered partitions $\mathcal{X}_1, \ldots, \mathcal{X}_k$ that have already been evaluated during step 2. When the algorithm is recursively applied to a part $X \in \mathcal{X}_{k+1}$ (step 2.a), one also retains the *complementary partition* $\overline{\mathcal{X}} = \mathcal{X}_{k+1} \backslash \{X\}$. Hence, within the "lower" calls, when a covered partition $\mathcal{Y} \in \mathfrak{C}(\{X\})$ is considered for branching, one first checks if $\overline{\mathcal{X}} \cup \mathcal{Y}$ does not refine any of the previously-evaluated partitions $\mathcal{X}_1, \ldots, \mathcal{X}_k$. If it does ($\exists i \leqslant k,\ \overline{\mathcal{X}} \cup \mathcal{Y} \in \mathfrak{R}(\mathcal{X}_i)$), one deduces that the branch has already been evaluated, and steps 2.a and 2.b may be avoided. Fig. 5 indicates the result of this improvement by crosses cutting the plain arrows.

Note however that this improvement should be implemented with the greatest care in order to be computationally efficient. In Section 5, where this generic algorithmic framework is applied to special versions of the SPP, the specific algebraic

structures resulting from the covering relation are fully known. We then show how one can directly generate, during step 2, the covered partitions that have not been evaluated yet, without actually recording them into memory. This way, the avoidance of redundant evaluations does not require additional ressources.

The branching method and the two improvements directly lead to the following algorithm:

A Generic Algorithm to Solve the SPP

Global Inputs:

c	a cost function;
$\mathcal{P}$	a set of admissible parts defining admissible partitions;
$\mathfrak{L}$	a set of locally-optimal admissible partitions of parts on which the algorithm has already been applied.

Local Inputs:

X	an admissible part;
$\overline{\mathcal{X}}$	the complementary partition of X inherited from the "higher" call ($\overline{\mathcal{X}}$ is a partition of $\Omega \backslash X$);
$\mathfrak{D}$	the set of admissible partitions which refinements have already been evaluated during "higher" calls.

Local Output:

$\mathcal{X}^*$	a locally-optimal admissible partition of X.

- If the algorithm has already been applied to part X, return the locally-optimal partition recorded in $\mathfrak{L}$.
- Initialization: $\mathcal{X}^* \leftarrow \{\{X\}\}$ and $\mathfrak{D}' \leftarrow \mathfrak{D}$.
- For each $\mathcal{Y} \in \mathfrak{C}(\{X\})$ such that $\overline{\mathcal{X}} \cup \mathcal{Y}$ does not refine any partition in $\mathfrak{D}$, do the following:
  - For each part $Y \in \mathcal{Y}$, call the algorithm with local inputs $X \leftarrow Y$, $\overline{\mathcal{X}} \leftarrow \overline{\mathcal{X}} \cup \mathcal{Y} \backslash \{Y\}$, and $\mathfrak{D} \leftarrow \mathfrak{D}'$ to compute a locally-optimal partition $\mathcal{Y}_Y^* \in \mathfrak{P}^*(Y)$.
  - $\mathcal{Y}^* \leftarrow \bigcup_{Y \in \mathcal{Y}} \mathcal{Y}_Y^*$.
  - If $c(\mathcal{Y}^*) > c(\mathcal{X}^*)$, then $\mathcal{X}^* \leftarrow \mathcal{Y}^*$.
  - $\mathfrak{D}' \leftarrow \mathfrak{D}' \cup \{\mathcal{Y}\}$.
- Return $\mathcal{X}^*$ and record this result in $\mathfrak{L}$.

---

5 From the Generic Framework to Specialized Implementations

The above algorithm can *in theory* be applied to any set of admissible parts $\mathcal{P} \subset 2^{\Omega}$. In that sense, it provides a dynamic algorithm to solve the *general* SPP. However, a generic implementation would not be computationally-optimal for *special versions* of the SPP. Indeed, assuming that one explicitly knows the global structure of the partition set that is meant to be searched, one can adapt the algorithm and the data structures to the problem's specific algebraic structure. That is what we call *deriving a specialized implementation of the algorithm*. For example, in the general case, there are 2^n possible admissible parts (where n is the size of Ω), so one needs at least n bits to identify one of them (*e.g.*, a binary string of size n indicating which individuals are contained in the identified part). In the case of the OSPP, the admissible parts are the $\frac{1}{2}\, n\, (n-1)$ intervals of Ω, so one roughly needs $2\, \log n$ bits to identify one of them (*e.g.*, two integers indicating the indexes of the interval extrema). This need for dedicated data structures also holds when representing admissible partitions, computing covered partitions, and avoiding redundant evaluations.

Hence, the generic algorithm should be used as a starting point to build specialized ones. This section makes the specialization process explicit for the HSPP and the OSPP by presenting, in both cases, the execution of the generic algorithm, the data structures used for the implementation, the pseudocode of the specialized algorithm, and the resulting computational complexity: *linear* in case of the HSPP and *quadratic* in case of the OSPP, thus meeting the results presented in Section 3. These two algorithms have been implemented in C++ within the visualization tool PajeNG for the spatial aggregation of execution traces of distributed computing systems [30] and within the Ocelotl module of the FrameSoC platform for the temporal aggregation of traces of embedded systems [20, 40]. Please refer to the GitHub repositories of Schnorr [51] and Dosimont [19] for more details regarding the implementation of the two specialized algorithms presented in this section.

5.1 Solving the HSPP

As it has been previously defined, the HSPP arises when the set of admissible parts $\mathcal{P}$ forms a *hierarchy* (see Subsection 3.3 for the formal definition). A $O(n)$ optimal algorithm, based on a depth-first search of the tree representing the hierarchy, has been proposed in independent work [28, 30, 43]. We show in this subsection how this result can be recovered as a special case of our generic framework.

Execution of the Generic Algorithm. An example of execution of the generic algorithm in the case of a 3-levels hierarchy is presented in Fig 6. For each admissible part $X \in \mathcal{P}$, the corresponding maximal partition $\{X\}$ covers only one admissible partition, that is the set of children of the node X in the tree representing the hierarchy: $\mathfrak{C}(X) = \{Y \in \mathcal{P} \setminus Y \subsetneq X, \nexists Z \in \mathcal{P}, Y \subsetneq Z \subsetneq X\}$. Hence, the branching of the search space proposed in Eq. 4 is not redundant: $\mathfrak{P}(X) = \{\{X\}\} \cup \mathfrak{R}(\mathfrak{C}(X))$. The resulting algorithm is then a simple recursive procedure consisting in a depth-first search of the tree (see Fig. 6): a recursive execution on each $Y \in \mathfrak{C}(X)$ (step 2.a), the building of the resulting locally-optimal partition $\bigcup_{Y \in \mathfrak{C}(X)} \mathcal{Y}^*_Y$ (step 2.b), and the comparison of its cost with the one of the maximal partition $\{X\}$ (step 3). The algorithm is naturally called only once per admissible part (no need for memoization, no stars on dashed lines in Fig. 6) and the branching of the partition set is never redundant (no crosses on plain arrows).

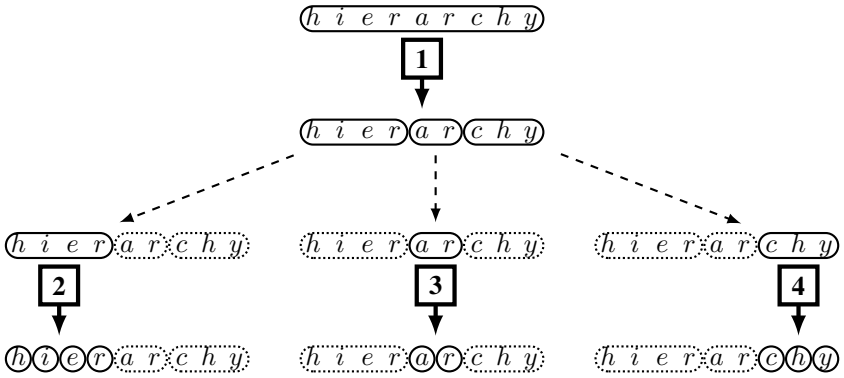

Figure 6: Execution trace of the optimal partitioning algorithm in the case of a hierarchical population of size 9 (see Fig. 1). Plain numbered arrows represent branchings (step 2). Dashed arrows represent recursive calls (step 2.a).

Data Structures and Implementation. The hierarchy $\mathcal{P}$ is implemented by a tree data structure. Each node represents an admissible part $X \in \mathcal{P}$ and has three labels instantiated by the algorithm:

- $cost$ stores the cost $c(X)$ of the corresponding part;
- $optimalCost$ stores the cost $c(\mathcal{X}^*)$ of a locally-optimal partition of X;
- $optimalCut$ is a Boolean value that is *true* if and only if the maximal partition $\{X\}$ is optimal among the admissible partitions of X. It thus provides a *cut* of the tree which represents an optimal admissible partition.

The pseudocode of the algorithm is given below (see Algorithm 1). The depth-first search computes the $optimalCost$ and the $optimalCut$ of each node according to its $cost$ and to the sum of the $optimalCost$ of its children. After the execution, the optimal partition of Ω is the union of the higher nodes in the tree such that $optimalCut = true$.

Algorithm 1 for the HSPP

Require: A tree with a label $cost$ on each node representing the cost of the corresponding admissible part.

Ensure: Each node of the tree has a Boolean label $optimalCut$ representing an optimal partition ($optimalCut = true$ if and only if the maximal partition is optimal for this node).

```
procedure SOLVEHSPP(node)
    if node has no child then
        node.optimalCost ← node.cost
        node.optimalCut ← true
    else
        MCost ← node.cost
        μCost ← 0
        for each child of node do
            SOLVEHSPP(child)
            μCost ← μCost + child.optimalCost
        end for
        node.optimalCost ← max(μCost, MCost)
        node.optimalCut ← (μCost < MCost)
    end if
end procedure
```

Linear Complexity. Since the recording of the three labels needs a constant memory space for each node, the space complexity of this algorithm is bounded by the size of the data tree representing the hierarchy. As it contains between $n+1$ (only the root and the leaves) and $2n-1$ nodes (in the case of a complete binary tree), the space complexity is *linear*. The operations needed to instantiate and exploit these labels are all achieved in constant time. Henceforth, the time complexity is the one of a depth-first search and is also *linear*, meeting the results of past algorithms dedicated to the HSPP (see Subsection 3.3).

5.2 Solving the OSPP

As it has been previously defined, the OSPP arises when the admissible parts are the intervals of Ω induced by a total order $<$ (see Subsection 3.4 for the formal definition). A $O(n^2)$ dynamic programming algorithm has been proposed in independent work to solve this problem [2, 26, 46, 53]. Here again, we show how this result can be recovered as a special case of our generic framework.

Execution of the Generic Algorithm. An example of execution of the generic algorithm for an ordered population of size 4 has been given in Fig. 5. Given a population Ω of n ordered individuals $x_1 < \ldots < x_n$, for each interval $[\![x_i, x_j]\!] = \{x_i, \ldots, x_j\}$ with $1 \leqslant i \leqslant j \leqslant n$, the admissible partitions covered by the maximal partition $\{[\![x_i, x_j]\!]\}$ are the couples of subintervals $\{[\![x_i, x_k]\!], [\![x_{k+1}, x_j]\!]\}$ with $i \leqslant k < j$. In the following, for the sake of conciseness, we simply mark such an interval $[i, j]$ and its covered partitions $[i, k][k+1, j]$. We thus have the following branching: $\mathfrak{C}([i, j]) = \{[i][i+1, j], \ldots, [i, j-1][j]\}$.

The generic algorithm is applied to $\Omega = [1, n]$. Let us assume that the covered partitions are evaluated (step 2) in the following order: $[1, n-1][n]$, $[1, n-2][n-1, n]$, $\ldots$, $[1][2, n]$ (see for example arrows **1**, **6** and **9** in Fig. 5). The covered partition $[1, n-1][n]$ is evaluated first (**1**). The algorithm is thus recursively applied (step 2.a) on part $[1, n-1]$ (**2**), then on part $[1, n-2]$ (**3**), and so on, until locally-optimal partitions of parts $[1], [1, 2], \ldots, [1, n-1]$ have been computed and recorded. All that remains is the computation of an optimal partition of part $[1, n]$. For the k^{th} evaluation, with $1 < k < n$, the covered partition $[1, n-k][n-k+1, n]$ has to be evaluated (for example **6**) knowing that the covered partitions $\{[1, n-i][n-i+1, n]\}_{1 \leqslant i < k}$ have already been evaluated along with their refined partitions. The algorithm is recursively applied to parts $[1, n-k]$ and $[n-k+1, n]$ (step 2.a):

- Since the algorithm has already been applied to part $[1, n-k]$ during the first evaluation, the optimal partition is simply read from memory (see cross below (1 2) in Fig. 5).
- Regarding part $[n-k+1, n]$, all covered partitions are now considered for evaluation. However, since $[n-k+1, n-i][n-i+1, n]$ refines $[1, n-i][n-i+1, n]$ for all $1 \leqslant i < k$, each covered partition has already been evaluated during the previous evaluations. Hence, steps 2.a and 2.b may be avoided (see star on arrow **8** in Fig. 5) and the algorithm uses the maximal partition $[n-k+1, n]$.

To sum up, in order to compute an optimal partition $\mathcal{X}^*_{[1,n]}$, the generic algorithm recursively computes locally-optimal partitions $\mathcal{X}^*_{[1]}, \mathcal{X}^*_{[1,2]}, \ldots, \mathcal{X}^*_{[1,n-1]}$. Then, it

exploits the results to compare partitions $\mathcal{X}^*_{[1]} \cup \{[2,n]\}, \ldots, \mathcal{X}^*_{[1,n-1]} \cup \{[n]\}$ and it returns one that has the highest cost.

Data Structures and Implementation. Given the population $\Omega = [1,n]$, each admissible part $[i,j] \in \mathcal{P}$ is represented by a couple of integer (i,j), with $1 \leqslant i \leqslant j \leqslant n$. The costs of admissible parts are recorded in a $n \times n$ upper triangular matrix $cost$. Each cell $cost[i,j]$, with $1 \leqslant i \leqslant j \leqslant n$, gives the cost $c([i,j])$ of the corresponding part. Optimal partitions are encoded in a vector $optimalCut$ containing n integers such that, for all $1 \leqslant j \leqslant n$, $optimalCut[j]$ is the indice of the first individual of the last part of an optimal partition of $[1,j]$. Hence, $optimalCut[n] = k$ indicates that part $[k,n]$ is in the optimal partition of $[1,n]$ and, if $k > 1$, then $optimalCut[k-1]$ again indicates the first individual of the last part of an optimal partition of $[1,k-1]$, and so on. The optimal partition of $[1,n]$ thus consists in a sequence of indices $k_1, \ldots, k_m$ recorded in $optimalCut$ and indicating the m individuals where the population is divided: $[1,k_1-1][k_1,k_2-1]\ldots[k_m,n]$. The costs of these optimal partitions are recorded in a vector $optimalCost$ of size n. Each cell $optimalCost[j]$, with $1 \leqslant j \leqslant n$, gives the cost of the optimal partitions of part $[1,j]$. The algorithm iteratively runs through the triangular matrix to build the two vectors and thus computes an optimal admissible partition of Ω. The pseudocode of the algorithm is given below (see Algorithm 2).

Algorithm 2 for the OSPP

Require: A matrix $cost$ recording the costs of intervals.
Ensure: The vector $optimalCut$ represents an optimal partition (see text above).

```
procedure SOLVEOSPP(j)
    if j > 1 then
        SOLVEOSPP(j − 1)
    end if
    optimalCost[j] ← cost[1, j]
    optimalCut[j] ← 1
    for cut ∈ ⟦2, j⟧ do
        μCost ← optimalCost[cut − 1] + cost[cut, j]
        if μCost > optimalCost[j] then
            optimalCost[j] ← μCost
            optimalCut[j] ← cut
        end if
    end for
end procedure
```

Quadratic Complexity. For a population of size n, the upper triangular matrix contains $\frac{1}{2}n(n+1)$ values and the two vectors each contains n integers. Hence, the space complexity is *quadratic*. In the proposed implementation, for each part $[1, j]$ with $1 \leqslant j \leqslant n$, the algorithm performs $j-1$ comparisons to identify the optimal partitions among the covering ones. Hence, overall, $(n-1)(n-2)/2$ comparisons are performed and the time complexity is also *quadratic*. This result meets the ones of the previous algorithms that have been developed for the OSPP (see Subsection 3.4).

6 Generalization and Limitation of the Framework

The SPP, as formalized in Subsection 3.1 and in most of the work herein referenced, is usually defined "at the part level". The set of admissible *parts* is an input of the problem from which the set of admissible *partitions* (*i.e.*, the search space) is straightforwardly generated; and a cost is associated to each *parts* from which the cost of *partitions* (*i.e.*, the objective function) is simply derived by summation. However, the expression of costs and admissibility constraints "at the partition level" might be required to model more complex problems where the feasibility and the quality of partitions cannot be simply derived from the feasibility and the quality of their parts. In this section, we present such cases where "the part level" is not longer sufficient to model the problem. Because the principle of optimality we introduced in Subsection 4.1 does not stand in these cases, we also discuss the strong limitations of the dynamic programming approach when dealing with this more general class of optimization problems (6.1 and 6.2). We also show in the last subsection how the algorithmic framework proposed in this paper can easily be improved to find not one, but all optimal partitions of a given instance (6.3).

6.1 Generalization to Decomposable Cost Functions

Most of the work related to the SPP – including this one – is interested in the optimization of an *additively decomposable* objective: $\forall \mathcal{X} \in \mathfrak{P},\ c(\mathcal{X}) = \sum_{X \in \mathcal{X}} c(X)$. This property is also referred to in the literature as the *sum property* [16], *block-additivity* [26], or simply *additivity* [43], and it has been proved to be satisfied by numerous objective functions such as measures based on distances between individuals (*e.g.*, sum/max of diameters/splits clustering [42]), quality measures for graph clustering (*e.g.*, modularity and performance [43]), classical measures from information theory (*e.g.*, Shannon entropy, Kullback-Leibler divergence [28]), and other probabilistic measures for model selection [26]. To go further, the annexe of this paper shows that the principle of optimality (Subsection 4.1) and its con-

verse (Subsection 6.3 below) actually hold for any objective that is *decomposable* by a *monotone* operator: $\exists\,\mathrm{op} : \mathbb{R}^* \to \mathbb{R},\ \forall \mathcal{X} \in \mathfrak{P},\ c(\mathcal{X}) = \mathrm{op}((c(X))_{X \in \mathcal{X}})$ and $x_i < x_i' \ \Rightarrow\ \mathrm{op}(x_1, \ldots, x_i, \ldots, x_n) < \mathrm{op}(x_1, \ldots, x_i', \ldots, x_n)$. Hence, our framework can be straightforwardly applied to minimize, for example, the *product* of positive costs: $\forall X \in \mathcal{P}, c(X) \geqslant 0$ and $c(\mathcal{X}) = \prod_{X \in \mathcal{X}} c(X)$.

However, a more general version of the SPP could be interested in optimizing an objective $c : \mathfrak{P} \to \mathbb{R}$ that is defined at the partition level and that might not be decomposable at all: the cost of a partition is not entirely determined by the cost of its parts. This is for example the case with the *lumping problem* [41], where the state space of a dynamical process needs to be partitioned and aggregated such that the resulting macro-dynamics satisfy some *closure property* (*e.g.*, informational closure, observational commutativity, Markovianity). In this setting, the quality of an aggregate strongly depends on the way other parts of the system are themselves aggregated. Hence, parts cannot be evaluated independently from the partition in which they appear and, as a consequence, the principle of optimality might not hold. Hence, no optimal substructure possibly helps to exploit the feasibility constraints and to design tractable algorithms. In this case, one surely needs to cling to heuristic or approximation algorithms to solve this more general SPP (see Subsection 2.2.1).

6.2 Defining Structural Constraints at the Partition Level

The classical SPP allows to model all kinds of constraints related to the structure of admissible parts, but it cannot express constraints regarding the structure of admissible partitions. Yet, much work is interested in computing optimal partitions with bounded size, thus imposing a constraint on the search space that cannot be expressed as constraints on the set of admissible parts: $\forall \mathcal{X} \in \mathfrak{P},\ |\mathcal{X}| < k$ [12] or $|\mathcal{X}| = k$ [9, 11, 22, 35, 36, 38, 42]. For another example, in the case of the APP (see Subsection 3.5), constraints are sometimes related to partition properties in addition to the rectangular shape of admissible parts [7, 35]: *hierarchical partitioning* requires a nested partitioning where each rectangular subset, beginning from the whole array, can only be partitioned into two subsets (no complex intertwined patterns can arise this way); $p \times q$ *partitioning* is even more specific since it requires that the sides of all rectangles are aligned with each others (the grid is actually partitioned by complete *guillotine cuts* in the vertical and horizontal axes).

Such constraints cannot be expressed within the classical SPP since, in this more general case, the set of admissible partitions does not fit with the set of partitions that are generated from the set of admissible parts. One problem when dealing with such partition constraints is that the branching scheme that is proposed in Section 4 cannot be used properly. Indeed, this scheme assumes that the admis-

sibility of a part is defined independently from the way the rest of the population is partitioned. On the contrary, the partition constraints cannot be preserved while independently looking for locally-optimal admissible partitions in each branch of the search space. As for the generalization to objectives defined at the partition level, one also needs to cling in this case to heuristic or approximation algorithms to solve this more general SPP.

6.3 Computing the Whole Set of Optimal Partitions

One might be interested in computing not only one, but all optimal partitions:

$$\mathfrak{P}^* \;=\; \arg\min_{\mathcal{X}\in\mathfrak{P}} c(\mathcal{X}).$$

Our generic algorithmic framework can easily be adapted by considering the converse of the principle of optimality. Intuitively, *a partition is potentially optimal only if its subpartitions also are*.

Theorem 2. *Let Ω be a population and c be an additive cost function defining partition optimality. For any partition $\mathcal{Y}$ of Ω, if a partition is optimal among the refinements of $\mathcal{Y}$, then it is the the union of locally-optimal partitions of the parts of $\mathcal{Y}$:*

$$\left(\bigcup_{Y\in\mathcal{Y}} \mathcal{Y}^*_Y\right) \in \mathfrak{R}^*(\mathcal{Y}) \quad\Rightarrow\quad \forall Y \in \mathcal{Y},\ \mathcal{Y}^*_Y \in \mathfrak{P}^*(Y) \tag{6}$$

Proof. *The proof of this theorem is given in annexe.*

Hence, the set of optimal partitions that refine a given partition is the Cartesian product of the sets of locally-optimal partitions of the parts of that given partition: $\mathfrak{R}^*(\mathcal{Y}) = \times_{Y\in\mathcal{Y}} \mathfrak{P}^*(Y)$, and Eq. 5 can be replaced by the following formula:

$$\mathfrak{P}^*(X) \;=\; \underset{\mathcal{X}\in\{\{X\}\}\cup\left(\bigcup_{\mathcal{Y}\in\mathfrak{C}(\{X\})}\left\{\times_{Y\in\mathcal{Y}}\mathfrak{P}^*(Y)\right\}\right)}{\arg\min} c(\mathcal{X}) \tag{7}$$

This leads to an algorithm very similar to the one presented in Section 4, except that one needs to record *sets of partitions* and compute their *Cartesian product*, instead of simply recording *partitions* and computing their *union*. Note that, in this setting, the time and space complexity of the resulting specialized implementations may no longer be tractable since, in the worst case, all partitions are optimal and the output set $\mathfrak{P}^*(X)$ hence has an exponential size. However, in practice, one might often assume that – except in extremal cases – the number of optimal partitions is bounded and very small (see for example [28]).

7 Conclusion and Perspectives

By exploiting strong assumptions regarding the global structure of the search space, dozens of problems have been modeled as tractable versions of the SPP. The algorithmic framework that we propose in this paper provides a unified dynamic programming approach to design such computationally-efficient optimal algorithms by exploiting the algebraic properties of such structures. This last section gives some application and research perspectives for this framework.

7.1 Applying the Generic Framework to Other Versions of the SPP

We have shown how the algorithmic framework might be easily applied to solve two well-known versions of the SPP, that is the HSPP and the OSPP. Section 5 made the specialization steps explicit: (1) formalization of the admissible parts and the resulting admissible partitions, (2) analysis of the generic algorithm execution, (3) design of data structures that fits with the induced algebraic structure, and (4) dynamic programing of the corresponding specialized algorithm. By following these steps, our programming method can be applied to numerous other versions of the SPP that might model interesting spatiotemporal structures: *e.g.*, partitioning graphs into connected components [7] or into spanning trees [31], partitioning partially ordered sets such as *interaction diagrams* representing causal relations [29], partitioning the state space of dynamical processes, also known as the *lumping problem* [41], partitioning multidimensional populations mixing spatial and temporal constraints, such as the Cartesian product of a total order and a hierarchy structure [20]. To the best of our knowledge, these special versions of the SPP have not been solved yet, and we believe that they might all benefit from the generic approach presented in this paper.

Another interesting research perspective is the following. For any such new version of the SPP, the computational complexity of the corresponding specialized implementation will be bounded from below by the number of admissible parts. Indeed, any algorithm should at least scan the input, that is the costs associated to each admissible part (or at least compute such costs from individual attributes). For both versions addressed in Section 5, the resulting specialized algorithms achieve such lower bounds: $O(n)$ for the HSPP and $O(n^2)$ for the OSPP. However, this is not always the case. *E.g.*, for a cyclic order, the number of admissible parts is $O(n^2)$, but dynamic programming only provides a $O(n^3)$ algorithm (see [46] and Table 1). This leads to interesting the following research questions: *For which versions of the SPP does the framework provide an optimization algorithm which complexity is bounded from below by the number of admissible parts?* Such version are hence linearly solvable with respect to the size of the instance $|\mathcal{P}|$. *If not, is*

it possible to do better, or does the generic framework provide a computationally-optimal algorithm? In that case, *which algebraic properties of the search space allow to identify problems that can be linearly solved regarding the size of the input?*

7.2 Enhancement Perspectives for the Generic Algorithmic Framework

Future work may build on the large literature regarding algorithms for special versions the SPP to enhance the generic framework proposed in this paper. Here are some research perspectives in that direction.

- In many real-world applications, costs are actually sparsely defined within the set of admissible parts. This is for example the case in the *winner determination problem* when no bid has been provided for many however-authorized combinations of assets, thus resulting in "holes" in the algebraic structure [48, 50]. Algorithms exploiting this *sparsity* of the cost function have been proposed to provide more efficient optimization procedures (for example in the case of the OSPP [48]). Generalizing this strategy to enhance our framework would consist in assuming infinite cost for such "holes" and in avoiding branches of the decomposition where they appear, thus speeding up the resulting specialized algorithms.

- *Pruning* consists in identifying subspaces of the search space that have no potential of containing any optimal solution. An adapted search space representation have been proposed in the case of the CSPP to identify independent subspaces for which one can easily identify upper and lower bounds [45,49]. Branches of the decomposition that lies outside of such bounds are then safully avoided. One could build on the ideas and results of Sandholm *et al.* in the case of the CSPP [49] to generalize this pruning approach to the generic framework.

- An *anytime* algorithm based on integer-partition has also been proposed in the case of the CSPP for the *coalition structure generation problem* [44, 45]. It consists in quickly generating a suboptimal solution with bound guarantees by evaluating a reasonable subspace of the search space, then slowly improving this solution and establishing better bounds by evaluating other branches of the decomposition. The algorithm thus always provides a solution if stopped before termination, but it might, in the worst case, search the entire space, resulting in a $O(n^n)$ time complexity in the case of the CSPP. Once again, by building on the results of Sandholm *et al.* [49], one

might identify in a general way to identify subspaces that provide reasonable bounds and to provide a generic framework to build specialized anytime algorithms.

- Because of the optimal substructure of the SPP and the branching we proposed in Section 4, the computational approach presented in this paper is easily *parallelizable*. For example, the depth-first search of the hierarchy that solves the HSPP (see Subsection 5.1) can be performed in sublinear time with a linear number of processors [23]. In the case of the *coalition structure generation problem* [49] and the interaction-analysis of multiagent executions [29], it has also been proposed to distribute the optimization process among agents and coalitions, so that they share the burden of computation. These examples encourage us to generalize our algorithmic framework, exploiting the dynamic approach as well as classical distributed computation tools in order to build specialized parallel algorithms for versions of the SPP.

Annexe

In this annexe, we prove the principle of optimality (Theorem 1 in Subsection 4.1) and its converse (Theorem 2 in Subsection 6.3). As mentioned in Subsection 6.1, these two principles hold for any cost function that is *decomposable* according to a *monotone* operator. Hence, this proof is more general than the classical SPP considering only *additive* objectives.

Let c be a cost function which is decomposable by a monotone operator op (see 6.1 for details). The proof is given for any partition $\mathcal{Y} = \{Y_1, Y_2\}$ of Ω and can easily be generalized to any partition $\mathcal{Y} = \{Y_1, \ldots, Y_k\}$. Because of *decomposability*, for any couple of partitions $\mathcal{Y}_1 \in \mathfrak{P}(Y_1)$ and $\mathcal{Y}_2 \in \mathfrak{P}(Y_2)$, we have $c(\mathcal{Y}_1 \cup \mathcal{Y}_2) = \mathrm{op}(c(\mathcal{Y}_1), c(\mathcal{Y}_2))$, and because of *monotonicity*, if $c(\mathcal{Y}_1) < c(\mathcal{Y}'_1)$, then $\mathrm{op}(c(\mathcal{Y}_1), c(\mathcal{Y}_2)) < \mathrm{op}(c(\mathcal{Y}'_1), c(\mathcal{Y}_2))$.

Proof of the Principle of Optimality (Th. 1). *Let $\mathcal{Y}_1 \in \mathfrak{P}^*(Y_1)$ and $\mathcal{Y}_2 \in \mathfrak{P}^*(Y_2)$. Since op is monotone, for any partition $\mathcal{Y}'_1 \cup \mathcal{Y}'_2 \in \mathfrak{R}(\{Y_1, Y_2\})$, we have:*

$$\begin{aligned} c(\mathcal{Y}_1 \cup \mathcal{Y}_2) &= \mathrm{op}(c(\mathcal{Y}_1), c(\mathcal{Y}_2)) \\ &\leqslant \mathrm{op}(c(\mathcal{Y}'_1), c(\mathcal{Y}'_2)) = c(\mathcal{Y}'_1 \cup \mathcal{Y}'_2) \end{aligned}$$

Therefore, $\mathcal{Y}_1 \cup \mathcal{Y}_2 \in \mathfrak{R}^*(\{Y_1, Y_2\})$. □

Proof of its Converse (Th. 2). *Let $\mathcal{Y}_1 \cup \mathcal{Y}_2 \in \mathfrak{R}^*(\{Y_1, Y_2\})$. For any partition $\mathcal{Y}'_1 \cup \mathcal{Y}'_2 \in \mathfrak{R}(\{Y_1, Y_2\})$, we have $c(\mathcal{Y}_1 \cup \mathcal{Y}_2) \leqslant c(\mathcal{Y}'_1 \cup \mathcal{Y}'_2)$. By contradiction,*

let us assume that $\mathcal{Y}_1 \notin \mathfrak{P}^*(Y_1)$. *Hence, there is a partition* $\mathcal{Y}'_1 \in \mathfrak{P}(Y_1)$ *such that* $c(\mathcal{Y}'_1) < c(\mathcal{Y}_1)$. *Hence, since* op *is monotone, we have:*

$$\begin{aligned} c(\mathcal{Y}'_1 \cup \mathcal{Y}_2) &= \mathrm{op}(c(\mathcal{Y}'_1), c(\mathcal{Y}_2)) \\ &< \mathrm{op}(c(\mathcal{Y}_1), c(\mathcal{Y}_2)) = c(\mathcal{Y}_1 \cup \mathcal{Y}_2) \end{aligned}$$

Since $\mathcal{Y}'_1 \cup \mathcal{Y}_2 \in \mathfrak{R}(\{Y_1, Y_2\})$, *there is a contradiction. This also holds for* $\mathcal{Y}_2 \notin \mathfrak{P}^*(Y_2)$. *Therefore,* $\mathcal{Y}_1 \in \mathfrak{P}^*(Y_1)$ *and* $\mathcal{Y}_2 \in \mathfrak{P}^*(Y_2)$. □

Acknowledgment

This work was partially supported by the French *Agence Nationale de la Recherche* under grant agreement ANR-12-CORP-0009 (GEOMEDIA project) and by the European Commission's 7th Framework Programme under grant agreement #318723 (MatheMACS project).

References

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

[2] S. Anily and A. Federgruen. Structured Partitioning Problems. *Operations Research*, 39(1):130–149, January/February 1991.

[3] J. P. Arabeyre, J. Fearnley, F. C. Steiger, and W. Teather. The Airline Crew Scheduling Problem: A Survey. *Transportation Science*, 3(2):140–163, 1969.

[4] Y. Bachrach, P. Kohli, V. Kolmogorov, and M. Zadimoghaddam. Optimal Coalition Structure Generation in Cooperative Graph Games. In M. desJardins and M. L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 81–87. AAAI Press, July 2013.

[5] Y. Bachrach, O. Lev, S. Lovett, J. S. Rosenschein, and M. Zadimoghaddam. Cooperative Weakest Link Games. In A. Lomuscio, P. Scerri, A. Bazzam, and M. Huhns, editors, *Proceedings of the Thirteen International Conference on Autonomous Agents and Multiagent Systems (AAMAS'14)*, pages 589–596. IFAAMAS, May 2014.

[6] E. Balas and M. W. Padberg. Set Partitioning: A Survey. *SIAM Review*, 18(4):710–760, 1976.

[7] R. Becker, I. Lari, M. Lucertini, and B. Simeone. Max-Min Partitioning of Grid Graphs into Connected Components. *Networks*, 32(2):115–125, 1998.

[8] N. Bilal, P. Galinier, and F. Guibault. A New Formulation of the Set Covering Problem for Metaheuristic Approaches. *ISRN Operations Research*, 2013, 2013.

[9] A. Björklund, T. Husfeldt, and M. Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

[10] M. Boschetti, A. Mingozzi, and S. Ricciardelli. A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, 5(4):735–747, 2008.

[11] P. Brucker. On the Complexity of Clustering Problems. *Optimization and Operations Research*, 157:45–54, 1978.

[12] A. K. Chakravarty, J. B. Orlin, and U. G. Rothblum. A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment. *Operations Research*, 30(5):1018–1022, 1982.

[13] C. Chekuri and A. Ene. Approximation Algorithms for Submodular Multiway Partition. In *Proceedings of the 2011 IEEE Fifty-second Annual Symposium on Foundations of Computer Science (FOCS'11)*, pages 807–816, October 2011.

[14] P. Chu and J. Beasley. Constraint Handling in Genetic Algorithms: The Set Partitioning Problem. *Journal of Heuristics*, 4(4):323–357, 1998.

[15] B. Cloitre. Sequence A003095. In *The On-Line Encyclopedia of Integer Sequences*. `http://oeis.org/A003095`, 2002.

[16] I. Csiszár. Axiomatic Characterizations of Information Measures. *Entropy*, 10(3):261–273, 2008.

[17] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, (2002).

[18] M. Dom. Set Cover with Almost Consecutive Ones. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 832–834. Springer US, 2008.

[19] D. Dosimont. lpaggreg. In *GitHub Repository*. `https://github.com/dosimont/lpaggreg`, 2013.

[20] D. Dosimont, R. Lamarche-Perrin, L. M. Schnorr, G. Huard, and J.-M. Vincent. A Spatiotemporal Data Aggregation Technique for Performance Analysis of Large-scale Execution Traces. In *Proceedings of the 2014 IEEE International Conference on Cluster Computing (CLUSTER'14)*. IEEE, September 2014.

[21] F. Eisenbrand, N. Kakimura, T. Rothvoß, and L. Sanità. Set Covering with Ordered Replacement: Additive and Multiplicative Gaps. *Integer Programming and Combinatoral Optimization*, 6655:170–182, 2011.

[22] A. H. Farrahi, D.-T. Lee, and M. Sarrafzadeh. Two-Way and Multiway Partitioning of a Set of Intervals for Clique-Width Maximization. *Algorithmica*, 23(3):187–210, 1999.

[23] J. Freeman. Parallel Algorithms for Depth-First Search. Technical Report MS-CIS-91-71, University of Pennsylvania, Department of Computer and Information Science, 1991.

[24] K. Hoffman and M. Padberg. Set Covering, Packing and Partitioning Problems. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2348–2352. Springer US, 2001.

[25] K. L. Hoffman and T. K. Ralphs. Integer and Combinatorial Optimization. In S. I. Gass and M. C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 771–783. Springer US, 2013.

[26] B. Jackson, J.D. Scargle, D. Barnes, S. Arabhi, A. Alt, *et al.* An algorithm for optimal partitioning of data on an interval. *IEEE Signal Processing Letters*, 12(2):105–108, 2005.

[27] R. Lamarche-Perrin, Y. Demazeau, and J.-M. Vincent. A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem. In A. Andreou and G. A. Papadopoulos, editors, *Proceedings of the 2014 IEEE International Conference on Tools with Artificial Intelligence (ICTAI'14)*. IEEE Computer Society, 2014.

[28] R. Lamarche-Perrin, Y. Demazeau, and J.-M. Vincent. Building the Best Macroscopic Representations of Complex Multi-Agent Systems. In *Transactions on Computational Collective Intelligence*, volume 15 of *LNCS 8670*, pages 1–27. Springer-Verlag Berlin, Heidelberg, 2014.

[29] R. Lamarche-Perrin, Y. Demazeau, and J.-M. Vincent. Macroscopic Observation of Large-scale Multi-agent Systems. In R. Prudencio and P. E. Santos,

editors, *Proceedings of the 2014 Brazilian Conference on Intelligent Systems (BRACIS'14)*, October 2014.

[30] R. Lamarche-Perrin, L. M. Schnorr, J.-M. Vincent, and Y. Demazeau. Evaluating Trace Aggregation for Performance Visualization of Large Distributed Systems. In T. M. Aamodt and B. C. Lee, editors, *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14)*, pages 139–140. IEEE Computer Society, March 2014.

[31] D. Lehmann, R. Müller, and T. Sandholm. The Winner Determination Problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, pages 297–317. MIT Press, 2006.

[32] C.-H. M. Lin and H. M. Salkin. An Efficient Algorithm for the Complete Set Partitioning Problem. *Discrete Applied Mathematics*, 6(2):149–156, 1983.

[33] G. McGarvey. Sequence A135361. In *The On-Line Encyclopedia of Integer Sequences*. `http://oeis.org/A135361`, 2007.

[34] S. Mecke and D. Wagner. Solving Geometric Covering Problems by Data Reduction. In S. Albers and T. Radzik, editors, *Algorithms – ESA 2004*, volume 3221 of *Lecture Notes in Computer Science*, pages 760–771. Springer Berlin Heidelberg, 2004.

[35] A. Mingozzi and S. Morigi. Partitioning a matrix with non-guillotine cuts to minimize the maximum cost. *Discrete Applied Mathematics*, 116(3):243–260, 2002.

[36] M. Minoux. A class of combinatorial problems with polynomially solvable large scale set covering/partitioning relaxations. *Revue française d'automatique, d'informatique et de recherche oprationnelle*, 21(2):105–136, 1987.

[37] R. Müller. Tractable Cases of the Winner Determination Problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, pages 319–336. MIT Press, 2006.

[38] S. Muthukrishnan and T. Suel. Approximation algorithms for array partitioning problems. *Journal of Algorithms*, 54(1):85–104, 2005.

[39] N. Nisan. Bidding and Allocation in Combinatorial Auctions. In *Proceedings of the Second ACM Conference on Electronic Commerce (EC'00)*, pages 1–12, New York, NY, USA, 2000. ACM.

[40] G. Pagano, D. Dosimont, G. Huard, V. Marangozova-Martin, and J.-M. Vincent. Trace Management and Analysis for Embedded Systems. In *Proceedings of the 7th International Symposium on Embedded Multicore SoCs (MC-SoC'13)*, pages 119–122. IEEE Computer Society Press, 2013.

[41] O. Pfante, N. Bertschinger, E. Olbricht, N. Ay, and J. Jost. Comparision between Different Methods of Level Identification. *Advances in Complex Systems*, 2013.

[42] J. Pintér and G. Pesti. Set partition by globally optimized cluster seed points. *European J. of Operational Research*, 51:127–135, 1991.

[43] P. Pons and M. Latapy. Post-processing hierarchical community structures: Quality improvements and multi-scale view. *Theoretical Computer Science*, 412(8-10):892–900, 2011.

[44] T. Rahwan and N. R. Jennings. Coalition Structure Generation: Dynamic Programming Meets Anytime Optimisation. In *Proceedings of the Twenty-third Conference on Artificial Intelligence*, pages 156–161. AAAI, 2008.

[45] T. Rahwan, S. D. Ramchurn, N. R. Jennings, and A. Giovannucci. An Anytime Algorithm for Optimal Coalition Structure Generation. *Journal of Artificial Intelligence Research*, 34(1):521–567, January 2009.

[46] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally Manageable Combinational Auctions. *Management Science*, 44(8):1131–1147, August 1998.

[47] N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. *Discrete Optimization*, 1(2):215–228, 2004.

[48] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.

[49] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209–238, 1999.

[50] T. Sandholm and S. Suri. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145:33–58, 2003.

[51] L. Schnorr. PajeNG – Trace Visualization Tool. In *GitHub Repository*. `https://github.com/schnorr/pajeng`, 2012.

[52] A. Schöbel. Set covering problems with consecutive ones property. Technical report, Universität Kaiserslautern, 2004.

[53] R. Vidal. Optimal Partition of an Interval – The Discrete Version. In *Applied Simulated Annealing*, volume 396 of *LNEMS*, pages 291–312. Springer Berlin, Heidelberg, 1993.

[54] D. Yun Yeh. A Dynamic Programming Approach to the Complete Set Partitioning Problem. *BIT Numerical Mathematics*, 26(4):467–474, 1986.