Università degli studi di Pavia
Dipartimento di Matematica
Corso di Laurea Magistrale in Matematica

UNIVERSITÀ
DI PAVIA

The Cloven Travelling Salesman: a new Approach to the ATSP Integrality Gap Estimation

Tesi di Laurea Magistrale in Matematica

Relatore (Supervisor)
Prof. Stefano Gualandi

Correlatore (Co-Supervisor)
Dott. Ambrogio Maria Bernardelli

Tesi di Laurea di
Alessandro Sosso
Matricola 527974

Anno Accademico 2023-2024

Abstract

The Travelling Salesman Problem (TSP) is a long-established combinatorial optimization problem, with a classic integer linear programming formulation by Dantzig, Fulkerson, and Johnson [10]. Due to its NP-hardness, approximation algorithms are of key importance, and so is the research on the TSP linear programming relaxation known as the Subtour Elimination Problem (SEP). The study of the integrality gap of the metric TSP, measuring the approximation accuracy of SEP as a ratio between the exact solution of TSP and the SEP relaxation, remains an open question as no definitive upper bound on it is known. The computation of the gap upper bound is generally achieved by enumerating the SEP polytope vertices and is thus practicable only for very small TSP instances, though estimates are obtained by restricting to subsets of high-gap vertices such as half-integer vertices. In this work, an alternative method for estimating the asymmetric TSP integrality gap is presented, where half-integer candidate instances are generated and then checked with efficient and original algorithms for their feasibility and extremality in the ASEP polytope. An implementation of the method was able to exhaustively determine the integrality gap of half-integer instances for $n \leq 11$, proving in a fraction of the runtime that the best-known estimates in the literature (by Elliott-Magwood [13]) are in fact the highest gaps attainable over half-integers.

Contents

Introduction

The Travelling Salesman Problem (TSP) is a long-established combinatorial optimization problem consisting in the search for the shortest-length (lowest-cost) tour visiting every node in a given weighted graph. In its traditional interpretation, origin of its name, the TSP is imagined as the quest of a salesman in need of finding a path taking him to n different cities exactly once and returning to the starting city, while traveling the shortest possible distance. Depending on whether the underlying weighted graph is directed or undirected, the problem is known as respectively the Symmetric (STSP) or Asymmetric (ATSP) Travelling Salesman Problem. In both forms the TSP is NP-hard, while its associated decision problem is NP-complete.

The popularity of the TSP and its lengthy history come from its high degree of generality as well as its versatility, that both sensibly explain why many applications of the TSP have emerged in the most diverse fields and how various related problems have sprouted from it. The TSP has been a natural choice for modeling logistics, scheduling and routing problems [1, 24], has been employed in the optimization of printed circuit boards assembly [12, 15] and inspection [20], and in the optimal scheduling in a processor of pending tasks [33], to cite some. An overview of TSP applications is provided by [22].

Being a cornerstone of theoretical computer science and operations research, many algorithms have been developed over time to solve the TSP. Among these, a fundamental method is the one developed by Dantzig, Fulkerson and Johnson in [10], which introduced on a now classic formulation of the TSP as an integer linear problem. In their famous paper, the authors developed a cutting plane method for solving the problem which became the foundation for the following evolution of mixed-integer programming theory. Their framework was later improved on by key contributions of R. Gomory [17] and V. Chvátal [9] in the theory of cutting-planes, by the branch-and-bound method developed by A. H. Land and A. G. Doig [21], and finally by the branch-and-cut method introduced by M. Padberg and G. Rinaldi [25] as a way of merging the previous two approaches. The history of the TSP is examined in depth by A. Schrijver in [28].

Despite considerable achievements in improving the TSP solving algorithms, the inherent difficulty of finding an efficient P algorithm — which as we know is at least as difficult as the notoriously hard (and potentially impossible) task of proving that $\mathsf{P} = \mathsf{NP}$ — led many efforts to be devoted to finding algorithms that could provide approximate solutions while running in polynomial time. Examples of this are the greedy nearest neighbour algorithm, the k-opt or Lin-Kernighan heuristic and the ant colony optimization algorithm. In the context of the ILP formulation, an approximating algorithm can be directly defined by considering the linear relaxation of the TSP, that is the non-integer linear problem obtained by dropping the integrality requirement from the TSP formulation; we refer to the resulting problem as the Subtour Elimination Problem SEP.

A natural question that follows from the definition of any approximation algorithm is whether there are any guarantees on its accuracy, interpreted as the distance of the approximate solution from the exact solution. In particular we would be interested in finding a value η which bounds the quotient between the result of the approximate and the exact solution (thus making the method a so called η-approximating algorithm). For the nearest neighbour algorithm and the k-opt heuristic instances where the approximation is inaccurate by a factor proportional to n have been found by Frieze, Galbiati and Maffioli [14]. An interesting result by Papadimitriou and Vempala [26] shows that if $\mathsf{P} \neq \mathsf{NP}$ is true then for any approximation of the metric TSP it holds $\eta > \frac{220}{219}$ for the symmetric and $\eta > \frac{117}{116}$ for the asymmetric case.

When dealing with LP relaxations of ILP problems the measure of this accuracy is expressed by the integrality gap, defined as the ratio between the optimal value of the ILP problem and that of its relaxation. While the integrality gap of the TSP is known to be unbounded in the general case, as instances achieving any arbitrary gap can be constructed[1] [13, §4.2], when considering only metric weights (that is an assignment of weights which satisfies the triangular inequalities) the estimation of the integrality gap is an open question. For the metric STSP the integrality gap is proven to be at most $\frac{3}{2}$ [32, 29], but no instances with a gap of $\frac{4}{3}$ or greater have ever been found, making many believe that it is in fact the tightest bound on the integrality gap. This led Carr and Vempala [6] to conjecture that $\frac{4}{3}$ is the tightest bound on the integrality gap of the metric ATSP too. In this last asymmetric case no mathematical bound is known on the gap, but the discovery by Charikar, Goemans, and Karloff [7] of a family of weighted graph with integrality gap tending to 2 disproved Carr and Vempala's conjecture. Williamson [31] also proved that the integrality gap for every n is bounded by $\lceil \log_2 (n) \rceil$.

The computation of the integrality gap for small TSP instances has also been of widespread interest, as it undoubtedly provides an insight useful to the search for the gap upper bound. As the maximal integrality gap cannot be computed directly, since it would require computing the gap for infinite instances of the TSP, indirect approaches are to be employed. We rely on the strategy exhibited in [2], [5] and [13] where an auxiliary LP problem, derived from the dual of the SEP, is leveraged. Such auxiliary GAP problem takes as input a feasible solution to the SEP, and by calculating its optimum on all vertices of the SEP polytope for a given n the maximal value Gap_n of the integrality gap over all n-nodes TSP instances can be obtained.

With regards to the asymmetric TSP, P. Elliott-Magwood in [13] (and their previous paper with S. Boyd [4]) managed to compute Gap_n for $n \leq 7$ via exhaustive generation of the asymmetric SEP polytope vertices; for $n \geq 8$ however the number of vertices had become too high for their extensive computation to be feasible. The author therefore focused on the family of half-integer instances (those with only $0, 1/2, 1$ components), which in many literature results had exhibited the highest gaps, in an attempt to find a good lower bound on the value of Gap_n. As such, Elliott-Magwood was able to analyse and compute the integrality gap of all half-integer extreme points for $n = 8$ and $n = 9$, before the computation becoming unfeasible again. For $n \geq 10$ no form of enumeration was practicable, but the author nonetheless provided some lower bounds on the value of Gap_n by concentrating on only a few interesting instances. To the best of our knowledge, these are the strongest results known regarding the exact computation of the ATSP

[1] Sahni and Gonzalez [27] additionally proved that in this case a poly-time η-approximating algorithm cannot in general be found.

integrality gap over all ATSP instances and over half-integer instances.

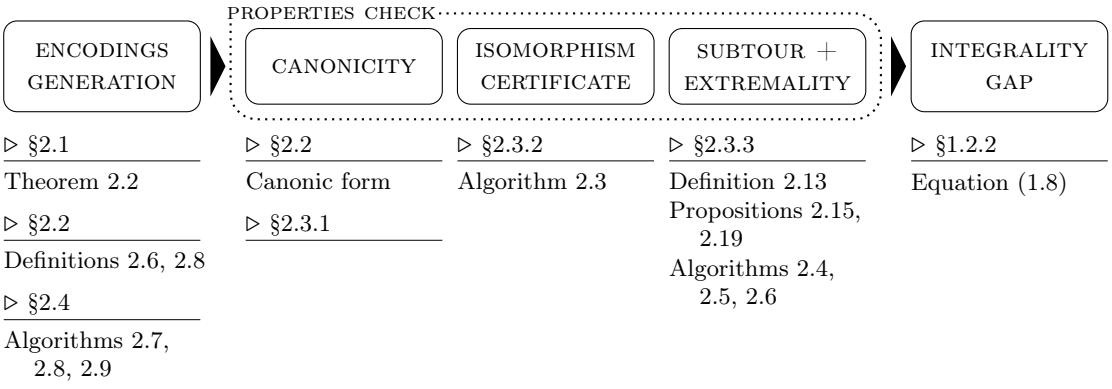
Subject of the thesis In this thesis, we propose a new approach to the generation of half-integer extreme points. Instead of generating vertices of the ASEP polytope and checking afterwards whether they are half-integer, we reverse the process by combinatorially generating a set of half-integer candidate points, checking afterwards whether they are feasible and extreme points of the ASEP polytope. The instances are generated by somewhat "cleaving" our poor salesman and asking for its two halves to be travelling along two vertex-disjoint cycle covers of the complete graph on n nodes, covers that are then combined to form a single half-integer candidate solution. In the main result of this work we show that the proposed procedure is exhaustive, proving that all half-integer extreme points can be written as a combination of two vertex-disjoint cycle covers.

With the objective of optimizing computation, much consideration has been devoted to removing isomorphic instances, by encoding in a smart way the underlying cycle covers and by checking the instances' isomorphism class. Furthermore, an original algorithm has been designed for detecting the extremality of candidate solutions, which was shown to perform significantly better than the direct check for rank maximality of the ASEP active constraints.

The consequent implementation of our approach was successful in finding new interesting results, as it was able to reproduce the results of [13] for $n \leq 9$ in a fraction of the runtime, and at the same time for $n = 10$ and $n = 11$ it proved that the gap lower bounds provided estimates by Elliott-Magwood are in fact the highest integrality gaps attainable over half-integer instances.

Outline of the text In Chapter 1 the preliminaries and notations on graph theory, the ATSP linear programming formulation, the integrality gap and its computation, and half-integer solutions are provided. Chapter 2 introduces our novel approach to the generation of half-integer solutions, presenting its theoretical justification as well as the structures and algorithms used in each step of the procedure. In Chapter 3 the results of our method's application are displayed, after a brief discussion on the details of the implementation and our efforts in its optimization. To conclude, future research directions and perspectives of the proposed technique are discussed.

A schematic overview of the proposed approach and its implementation is shown below, with relevant text sections, key definitions and results for each of its parts.



1. Preliminaries

As drafted, let us begin first and foremost with an overview of the necessary preliminary notions, from graph theory to half-integer solutions and their properties.

1.1 Graphs

Definition 1.1 (directed graph)**.** A *directed graph* $\mathcal{G} = (V, E)$ or *digraph* is a pair composed of a set of *vertices* V and a binary, non-reflexive relation E on V, whose elements are the *arcs* of the digraph. For any subset of arcs $E' \subseteq E$ we write the subgraph of $\mathcal{G}$ obtained by considering only the arcs in E' as $\mathcal{G}|_{E'} = (V, E')$.

For a graph on $|V| = n$ vertices we usually consider

$$V = V^n \coloneqq \{0, 1, \dots, n-1\}, \qquad E \subseteq E^n \coloneqq \{(u, v) \mid u, v \in V^n,\ u \neq v\}.$$

Since any digraph on n vertices is equivalent up to isomorphism to a digraph in this form, it is sufficient to restrict any general argument to these digraphs only. We denote a complete digraph on n vertices as $\mathcal{K}_n = (V^n, E^n)$, and for convenience we usually write arcs as uv in place of (u, v) in the remainder of the text.

Let us also define the following functions for any $S \subseteq V^n$

$$\begin{aligned} \delta^+_{\mathcal{G}}(S) &\coloneqq \{uv \in E \mid u \in S, v \notin S\} = \delta^-_{\mathcal{G}}(V \setminus S), & N^+_{\mathcal{G}}(S) &\coloneqq \textstyle\bigcup_{s \in S} \delta^+_{\mathcal{G}}(\{s\}), \\ \delta^-_{\mathcal{G}}(S) &\coloneqq \{uv \in E \mid u \notin S, v \in S\} = \delta^+_{\mathcal{G}}(V \setminus S), & N^-_{\mathcal{G}}(S) &\coloneqq \textstyle\bigcup_{s \in S} \delta^-_{\mathcal{G}}(\{s\}). \end{aligned}$$

The subscripts are dropped for simplicity when no ambiguity arises, and we write $\delta^+_{\mathcal{G}}(w)$ and $\delta^-_{\mathcal{G}}(w)$ in place of $\delta^+_{\mathcal{G}}(\{w\})$ and $\delta^-_{\mathcal{G}}(\{w\})$ respectively.

Definition 1.2. A graph is said to be *weighted* if a (real) weight or *cost* is assigned to each arc, or in other words if a function $c\colon E \to \mathbb{R}$, $c\colon e \mapsto c_e$ is given.

We focus on a particular family of weighted graphs for which interesting results, tighter than the general case, hold.

Definition 1.3 (metric graph)**.** A weighted graph is said to be *metric*[1] if all costs are non-negative $c_{uv} \geq 0$ and satisfy the triangle inequality $c_{uv} \leq c_{uw} + c_{wv}$ for all distinct $u, v, w \in V$. A metric graph is also *symmetric* when $c_{uv} = c_{vu}$ for all $uv \in E$.

[1]Since our definition in general does not ask for the weights to be symmetric $c_{uv} = c_{vu}$ and strictly positive $c_{uv} > 0$ for $u \neq v$, it would be more accurate to refer to them as *pseudoquasimetric* weights; for simplicity however we will still refer to them as *metric*, in accordance with previous works in the literature such as [13].

Weighted graphs in particular can generally be assumed to be complete by setting the cost of the missing arcs in an adequate way (usually 0, infinity or by using the triangle inequalities $c_{uv} = c_{uw} + c_{wv}$).

To conclude we need to define the concepts of cycle cover and of Hamiltonian cycle or tour, extensively used in the remainder of the text.

Definition 1.4 (vertex cycle cover)**.** A *vertex cycle cover* of a graph $\mathcal{G}$ is a subset $C \subseteq E$ of the arcs of $\mathcal{G}$ that can be written as union of cycles and contains all the vertices of $\mathcal{G}$. If no two cycles in a cover have a vertex in common, the cover is said to be *vertex-disjoint.*

Only vertex-disjoint cycle covers will be considered, and referred to as *cycle cover*s or simply *covers.*

Definition 1.5 (Hamiltonian cycle)**.** An *Hamiltonian cycle* or *tour* is a cycle that visits each vertex of a graph exactly once, or alternatively a cycle cover with only one cycle.

Any cycle/tour/cover C, or in general any subset of arcs, can be represented as a binary vector: having established an ordering on the arcs $E = (e_1, e_2, \dots, e_k)$, we write $\boldsymbol{x} = (x_{e_1}, x_{e_2}, \dots x_{e_k}) \in \{0,1\}^{|E|}$ with

$$x_{e_i} \coloneqq \begin{cases} 1 & \text{if } x_{e_i} \in C, \\ 0 & \text{if } x_{e_i} \notin C. \end{cases}$$

We call this $\boldsymbol{x}$ the *characteristic* or *incidence vector* of C and C the cycle/tour/cover associated with $\boldsymbol{x}$; when no confusion arises, we may refer directly to the cycle/tour/cover $\boldsymbol{x}$ in place of C. We notate as $\operatorname{supp}_E(\boldsymbol{x})$ the set of arcs e of E for which $x_e > 0$, that is to say

$$\operatorname{supp}_E(\boldsymbol{x}) \coloneqq \{e \in E \mid x_e > 0\}.$$

Hall's Marriage Theorem

A classic result by P. Hall [19], useful in proving Theorem 2.2, provides a necessary and sufficient condition for the existence of a perfect matching on bipartite graphs, defined as follows.

Definition 1.6 (bipartite graph)**.** A graph $\mathcal{G} = (V, E)$ is said to be *bipartite* if there exists two disjoint partitions V_1, V_2 of $V = V_1 \cup V_2$ such that all arcs in E connect vertices in V_1 to vertices in V_2 or vice versa, that is

$$E \subseteq (V_1 \times V_2) \cup (V_2 \times V_1).$$

Definition 1.7 (perfect matching)**.** A subset of arcs of $\mathcal{G} = (V, E)$ is known as a *matching* if any two of its elements do not have a common vertex. A matching in said to be *perfect* if all vertices in V are extremity of an arc of the matching.

Hall's marriage theorem, in its graph theoretic formulation, states that:

Theorem 1.8 (Hall's marriage theorem)**.** *There exists a perfect matching on a bipartite graph $\mathcal{G} = (V, E)$ with $V_1, V_2 \subset V$ as bipartite sets if and only if*

$$|S| \leq \left|N_{\mathcal{G}}^{+}(S)\right|, \qquad \forall\, S \in \mathscr{P}(V_1) \cup \mathscr{P}(V_2).$$

1.2 Travelling Salesman Problem

The classical formulation of the ATSP is as an integer linear programming problem (an extension to the asymmetric case of the fundamental model introduced by Dantzig, Fulkerson, and Johnson in [10]), where the variables represent the components of the characteristic vector of the tour, that is:

$$\text{minimize} \quad \sum_{uv \in E^n} c_{uv} x_{uv} \tag{1.1}$$

$$\text{subject to} \quad \sum_{uv \in \delta^+(w)} x_{uv} = 1 \quad \forall\, w \in V^n \qquad \text{(out-degree constraints)} \tag{1.2}$$

$$\sum_{uv \in \delta^-(w)} x_{uv} = 1 \quad \forall\, w \in V^n \qquad \text{(in-degree constraints)} \tag{1.3}$$

$$\sum_{uv \in \delta^+(S)} x_{uv} \geq 1 \quad \forall\, S \in \mathscr{S}^n \qquad \text{(subtour elimination constraints)} \tag{1.4}$$

$$x_{uv} \geq 0 \quad \forall\, uv \in E^n \qquad \text{(bound constraints)} \tag{1.5}$$

$$x_{uv} \in \mathbb{Z} \quad \forall\, uv \in E^n, \qquad \text{(integrality constraints)} \tag{1.6}$$

with $\mathscr{S}^n \coloneqq \{S \subset V^n \mid 2 \leq |S| \leq n-2\}$.

It is safe to assume that the graph underlying the formulation of this problem is a complete graph $\mathcal{K}_n$, since it is possible to insert any missing arc uv with a weight of infinity or in the case of a metric graph by assigning it the cost of the shortest path between u and v without changing the optimal solution. As a consequence it can be seen that a particular instance of the ATSP and its relative solution is entirely determined by the vector of arc costs $\boldsymbol{c} = (c_{0\,1}, c_{0\,2}, \ldots, c_{n\,n-1})$, allowing us to write it as $\text{ATSP}\,(\boldsymbol{c})$.

An improvement that can immediately be incorporated into the model of (1.1) concerns the subtour elimination constraints (1.4) and is a consequence of the following.

Proposition 1.9. *Let $S \in \mathscr{S}^n$ and $S^{\complement} \coloneqq V^n \setminus S$. Then*

$$\sum\nolimits_{uv \in \delta^+(S)} x_{uv} = \sum\nolimits_{uv \in \delta^+(S^{\complement})} x_{uv}.$$

Proof. Since the following decompositions hold

$$\delta^+(S) = \bigcup_{u \in S} \Big(\delta^+(u) \setminus \big\{ uv \,\big|\, v \in S \setminus \{u\} \big\} \Big) = \bigcup_{v \in S^{\complement}} \Big(\delta^-(v) \setminus \big\{ uv \,\big|\, u \in S^{\complement} \setminus \{v\} \big\} \Big),$$

we have from the degree constraints and with $\Delta \coloneqq \{uv \in S \times S \mid u \neq v\}$

$$\sum_{uv \in \delta^+(S)} 4x_{uv} = \sum_{u \in S} 4\Big(\sum_{v \in \delta^+(u)} 4x_{uv} - \sum_{v \in S \setminus \{u\}} 4x_{uv} \Big) = \sum_{u \in S} 4\Big(1 - \sum_{v \in S\{u\}} 4x_{uv} \Big) = |S| - \sum_{uv \in \Delta} 4x_{uv},$$

$$\sum_{uv \in \delta^+(S^{\complement})} 4x_{uv} = \sum_{v \in S} 4\Big(\sum_{u \in \delta^-(v)} 4x_{uv} - \sum_{u \in S \setminus \{v\}} 4x_{uv} \Big) = \sum_{v \in S} 4\Big(1 - \sum_{u \in S \setminus \{v\}} 4x_{uv} \Big) = |S| - \sum_{uv \in \Delta} 4x_{uv}.$$

□

This result implies that if a solution $\boldsymbol{x}$ satisfies the subtour elimination constraint for S it also does so for $S^{\complement}$, and when one constraint is satisfied with equality (making it an *active* constraint), the other is satisfied with equality as well. It is therefore possible for any fixed choice of $\overline{w} \in V$ to replace all of the constraints in (1.4) with

$$\sum_{uv \in \delta^+(S)} x_{uv} \geq 1 \quad \forall\, S \in \mathscr{S}^n_{\overline{w}} \coloneqq \{S \in \mathscr{S}^n \mid \overline{w} \in S\},$$

effectively halving the number of constraints.[2] Despite the reduction in the number of constraints, the resulting formulation is not stronger then the original one, as we are removing duplicate constraints and as such the feasible solutions space is unchanged.

1.2.1 Subtour Elimination Problem and Integrality Gap

As remarked in the introduction, the NP-hardness of the TSP boosted the development of approximating methods among which stands the algorithm given by the linear relaxation of (1.1), obtained by dropping the integrality constraints (1.6); we refer to the resulting LP problem as the Asymmetric Subtour Elimination Problem (ASEP), for which the simplex method and its counterparts consequently give a P solving algorithm. As it is the case for the ATSP, any instance and solution of the ASEP is determined by the vector of costs $\boldsymbol{c}$, therefore we analogously write it as $\text{ASEP}(\boldsymbol{c})$.

We notate as $\mathfrak{P}^n$ the polytope that constitutes the region of the feasible solutions to the ASEP for a (complete) digraph on n vertices, that is

$$\mathfrak{P}^n \coloneqq \left\{ \boldsymbol{x} \in \mathbb{R}^{|E^n|} \;\middle|\; (1.2), (1.3), (1.4), (1.5) \right\}.$$

Linear optimization theory shows that the solution to any ASEP instance is always an extreme point of $\mathfrak{P}^n$: we denote the (finite) set of extreme points of $\mathfrak{P}^n$ as $\mathfrak{X}^n$. As such

$$\text{ASEP}(\boldsymbol{c}) = \min_{\overline{\boldsymbol{x}} \in \mathfrak{X}^n} \boldsymbol{c} \cdot \overline{\boldsymbol{x}}. \tag{1.7}$$

We now would like to introduce the integrality gap of the ASEP relaxation of ATSP in order to study its approximation accuracy, the main topic of this thesis. In the setting of the ATSP with a non-negative cost $\boldsymbol{c} \neq \boldsymbol{0}$, this can be expressed as

$$\text{Gap}(\boldsymbol{c}) \coloneqq \frac{\text{ATSP}(\boldsymbol{c})}{\text{ASEP}(\boldsymbol{c})} \geq 1.$$

Knowing that the integrality gap is unbounded in the general case, we restrict our discussion to only metric ATSP weights. As such we introduce the following quantity, expressing the maximal value of the integrality gap over all n-nodes, metric, non-degenerate ATSP instances,

$$\text{Gap}_n \coloneqq \max \left\{ \text{Gap}(\boldsymbol{c}) \;\middle|\; \boldsymbol{c} \in \mathbb{R}^{|E^n|},\ \boldsymbol{c} \text{ metric},\ \boldsymbol{c} \neq \boldsymbol{0} \right\}.$$

[2] $|\mathscr{S}^n_{\overline{w}}| = 2^{n-1} - n - 1$ against $|\mathscr{S}^n| = 2^n - 2n - 2$.

1.2.2 Integrality Gap Computation

To compute Gap_n we follow the same indirect strategy used by [2], [5] and [13], which relies on the introduction of the auxiliary problem GAP. We start by observing that any possible value for Gap can be achieved by a choice of $\boldsymbol{c}$ for which $\mathrm{ATSP}(\boldsymbol{c}) = 1$.

Proposition 1.10. *For any metric cost $\boldsymbol{c} \in \mathbb{R}^{|E^n|}, \boldsymbol{c} \neq \boldsymbol{0}$ of the complete graph $\mathcal{K}_n$ there exists a metric cost $\widetilde{\boldsymbol{c}} \in \mathbb{R}^{|E^n|}$ such that $Gap(\boldsymbol{c}) = Gap(\widetilde{\boldsymbol{c}})$ and $ATSP(\widetilde{\boldsymbol{c}}) = 1$.*

Proof. First, it is easy to see that $\mathrm{ATSP}(\boldsymbol{c}) > 0$. If this were not the case, the shortest tour $\boldsymbol{x}$ would have a total cost of 0; but since $\boldsymbol{c}$ is metric, any arc uv not in the tour $\boldsymbol{x}$ would itself have cost $c_{uv} = 0$, since by the triangular inequalities it cannot be greater than the cost of the path in $\boldsymbol{x}$ between u and v; this contradicts the hypothesis of $\boldsymbol{c} \neq \boldsymbol{0}$.

The new cost is then simply obtained as $\widetilde{\boldsymbol{c}} \coloneqq \frac{\boldsymbol{c}}{\mathrm{ATSP}(\boldsymbol{c})}$, easily shown to be non-null and metric, for which immediately $\mathrm{ATSP}(\widetilde{\boldsymbol{c}}) = 1$ and

$$\mathrm{Gap}(\widetilde{\boldsymbol{c}}) = \frac{\mathrm{ATSP}(\widetilde{\boldsymbol{c}})}{\mathrm{ASEP}(\widetilde{\boldsymbol{c}})} = \frac{1}{\mathrm{ASEP}(\widetilde{\boldsymbol{c}})} = \frac{1}{\mathrm{ASEP}(\boldsymbol{c}) \,/\, \mathrm{ATSP}(\boldsymbol{c})} = \mathrm{Gap}(\boldsymbol{c}).$$

□

Therefore $\mathrm{Gap}_n = \max\left\{ \frac{1}{\mathrm{ASEP}(\boldsymbol{c})} \;\middle|\; \boldsymbol{c} \in \mathbb{R}^{|E^n|} \text{ metric}, \ \mathrm{ATSP}(\boldsymbol{c}) = 1 \right\}$, or equivalently the constraint $\mathrm{ATSP}(\boldsymbol{c})$ can be relaxed to $\mathrm{ATSP}(\boldsymbol{c}) \geq 1$. Given a tour T in $\mathcal{K}_n$ with weights $\boldsymbol{c}$ and its associated vector $\boldsymbol{x}$, let $\mathrm{cost}(T) \coloneqq \sum_{e \in E^n} c_e x_e$ be the total cost of the tour T. Asking for $\mathrm{ATSP}(\boldsymbol{c}) \geq 1$ becomes then equivalent to asking for $\mathrm{cost}(T) \geq 1$ for every tour T in $\mathcal{K}_n$. Thus we can rewrite

$$\frac{1}{\mathrm{Gap}_n} = \min\left\{ \mathrm{ASEP}(\boldsymbol{c}) \;\middle|\; \boldsymbol{c} \in \mathbb{R}^{|E^n|} \text{ metric}, \ \mathrm{cost}(T) \geq 1 \ \forall T \text{ tour in } \mathcal{K}_n \right\}.$$

From equation (1.7) it also follows

$$\begin{aligned}\frac{1}{\mathrm{Gap}_n} &= \min\left\{ \min_{\overline{\boldsymbol{x}} \in \mathfrak{X}^n} \boldsymbol{c} \cdot \overline{\boldsymbol{x}} \;\middle|\; \boldsymbol{c} \text{ metric}, \ \mathrm{cost}(T) \geq 1 \ \forall T \text{ tour in } \mathcal{K}_n \right\} = \\ &= \min_{\overline{\boldsymbol{x}} \in \mathfrak{X}^n} \left\{ \min\left\{ \boldsymbol{c} \cdot \overline{\boldsymbol{x}} \;\middle|\; \boldsymbol{c} \text{ metric}, \ \mathrm{cost}(T) \geq 1 \ \forall T \text{ tour in } \mathcal{K}_n, \ \arg\min \mathrm{ASEP}(\boldsymbol{c}) = \overline{\boldsymbol{x}} \right\} \right\}.\end{aligned}$$

For a given extreme point $\overline{\boldsymbol{x}} \in \mathfrak{X}^n$ of $\mathfrak{P}^n$, the minimization problem inside the external brackets is solved by the linear programming problem GAP which can be derived from the dual of ATSP and is defined by

$$\begin{aligned}
\text{minimize} \quad & \sum_{uv \in E^n} \overline{x}_{uv} c_{uv} && && (1.8)\\
\text{subject to} \quad & c_{uw} + c_{wv} - c_{uv} \geq 0 && \forall\, uv \in E^n, w \in V^n \setminus \{u, v\} \\
& \mathrm{cost}(T) \geq 1 && \forall\, T \text{ tour in } \mathcal{K}_n \\
& c_{uv} - y_u^{\mathrm{out}} - y_v^{\mathrm{in}} - \textstyle\sum_{S \in \mathscr{S}^n_{uv}} d_S \geq 0 && \forall\, uv \in E^n \setminus \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}}) \\
& c_{uv} - y_u^{\mathrm{out}} - y_v^{\mathrm{in}} - \textstyle\sum_{S \in \mathscr{S}^n_{uv}} d_S = 0 && \forall\, uv \in \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}}) \\
& c_{uv} \geq 0 && \forall\, uv \in E^n \\
& d_S \geq 0 && \forall\, S \in \mathscr{S}^n,
\end{aligned}$$

with $\mathscr{S}^n_{uv} \coloneqq \{S \in \mathscr{S}^n \mid uv \in \delta^+(S)\}$. To summarize, we thus have

$$\frac{1}{\mathrm{Gap}_n} = \min_{\overline{\boldsymbol{x}} \in \mathfrak{X}^n} \left\{ \mathrm{GAP}\left(\overline{\boldsymbol{x}}\right) \right\}.$$

Even though this new formulation makes computing the integrality gap possible, for relatively small values of n its actual calculation becomes computationally unfeasible due to the exponentially growing number of points in $\mathfrak{X}^n$. This adds up to the fact that the LP problem GAP has a large amount of constraints, whose number grows rapidly with n, making its resolution increasingly expensive.[3] As such the search for a reduction in the number of instances of $\mathfrak{X}^n$ that need to be considered, for effective methods to generate those instances, and for efficient ways to solve GAP is naturally prompted.

1.2.3 Structure of $\mathfrak{X}^n$ and Half-Integer Solutions

Since every point in $\mathfrak{X}^n$ is the solution to a system of linear equations with integer coefficients, it is immediate to see how any $\overline{\boldsymbol{x}} \in \mathfrak{X}^n$ has only rational components, that is $\mathfrak{X}^n \subset \mathbb{Q}^{|E^n|} \cap [0,1]^{|E^n|}$. This allows us to write any $\overline{\boldsymbol{x}} \in \mathfrak{X}^n$ as $\overline{\boldsymbol{x}} = \left(\frac{k_1}{\alpha}, \frac{k_2}{\alpha}, \ldots, \frac{k_{|E^n|}}{\alpha}\right)$ for some appropriate $\alpha \in \mathbb{N}^+, k_i \in \mathbb{N}, 0 \leq k_i \leq \alpha, i = \{1, \ldots, |E^n|\}$. Let us therefore define

$$\mathfrak{P}^n_\alpha \coloneqq \mathfrak{P}^n \cap \left\{0, \frac{1}{\alpha}, \ldots, 1\right\}^{|E^n|}, \qquad \mathfrak{X}^n_\alpha \coloneqq \mathfrak{X}^n \cap \left\{0, \frac{1}{\alpha}, \ldots, 1\right\}^{|E^n|} = \mathfrak{X}^n \cap \mathfrak{P}^n_\alpha,$$

for which $\mathfrak{X}^n = \bigcup_{\alpha=1}^{\infty} \mathfrak{X}^n_\alpha$ holds.

Any solution $\overline{\boldsymbol{x}} \in \mathfrak{X}^n_\alpha$ (and in general any vector $\overline{\boldsymbol{x}} \in \mathbb{R}^{|E^n|}$) can be seen as a arc-weighed graph obtained by taking the subgraph $\mathcal{K}_n|_{\mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})}$ and weighting every arc $e \in \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})$ with $\overline{x}_e$. We say that this is the graph *associated to* $\overline{\boldsymbol{x}}$, denoted with $\mathcal{X}^n_{\overline{\boldsymbol{x}}}$ from here onward. The introduction of the associated graph also allows us to define the concept of isomorphism in $\mathfrak{X}^n$.

Definition 1.11. Given any two $\overline{\boldsymbol{x}}_1, \overline{\boldsymbol{x}}_2 \in \mathfrak{X}^n$, or in general any two $\overline{\boldsymbol{x}}_1, \overline{\boldsymbol{x}}_2 \in \mathbb{R}^{|E^n|}$, we say that $\overline{\boldsymbol{x}}_1$ and $\overline{\boldsymbol{x}}_2$ are *isomorphic* if and only if there exists a weight-preserving graph isomorphism between $\mathcal{X}^n_{\overline{\boldsymbol{x}}_1}$ and $\mathcal{X}^n_{\overline{\boldsymbol{x}}_2}$.

Since by definition a graph isomorphism is a permutation of the graph vertices, thanks to the symmetry in the formulation of the GAP problem we can observe that the value of GAP is invariant over isomorphic instances of $\mathfrak{X}^n$. Thus, in order to evaluate Gap_n, it suffices to check the value of GAP for only one representative of each class of isomorphism.

Half-Integers

Among all points of $\mathfrak{X}^n$, the subset $\mathfrak{X}^n_2$ for $\alpha = 2$ empirically appears to have high integrality gaps (low GAP results), having achieved the highest gaps for those values of n that allowed for an exhaustive search of $\mathfrak{X}^n$. The elements of $\mathfrak{X}^n_2$ are known as *half-integer* extreme points, since all its elements $\overline{\boldsymbol{x}} \in \mathfrak{X}^n_2$ are made up of components in the set $\{0, 1/2, 1\}$. It is thus reasonable to conjecture that this trend holds for all n,

$$\mathrm{Gap}_n = \max_{\overline{\boldsymbol{x}} \in \mathfrak{X}^n} \left\{ \mathrm{GAP}\left(\overline{\boldsymbol{x}}\right)^{-1} \right\} \simeq \max_{\overline{\boldsymbol{x}} \in \mathfrak{X}^n_2} \left\{ \mathrm{GAP}\left(\overline{\boldsymbol{x}}\right)^{-1} \right\},$$

[3] For $n = 11$, GAP already has 2,156 variables and 3,629,900 constraints.

or if this were not the case, that the class $\mathfrak{X}_2^n$ would at least provide a tight lower bound on the value of $\mathrm{Gap}_n \geq \max_{\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n} \left\{ \textsc{gap}\left(\overline{\boldsymbol{x}}\right)^{-1} \right\}$.

To further restrict our search, it is also possible to avoid considering instances of $\mathfrak{X}^n$ that have one or more components with value 1. This comes as a consequence of a result by Elliott-Magwood [13, §4.6] which states that given any $\overline{\boldsymbol{x}} \in \mathfrak{X}^n$ with a 1-arc (say uv, that is $\overline{x}_{uv} = 1$), the point $\overline{\boldsymbol{x}}' \in \mathfrak{P}^{n-1}$ obtained by identifying u and v to a single new vertex w, namely

$$\overline{x}'_{ab} = \begin{cases} x_{vb} & \text{if } a = w, \\ x_{au} & \text{if } b = w, \\ x_{ab} & \text{otherwise,} \end{cases} \tag{1.9}$$

has a value of GAP no greater than that of $\overline{\boldsymbol{x}}$ (also $\overline{\boldsymbol{x}}'$ is still an extreme point, $\overline{\boldsymbol{x}}' \in \mathfrak{X}^{n-1}$),

$$\textsc{gap}\left(\overline{\boldsymbol{x}}\right) \geq \textsc{gap}\left(\overline{\boldsymbol{x}}'\right),$$

that is to say that the removal of 1-arcs never decreases the integrality gap.

Let us refer to half-integer extreme points with only $0, 1/2$-components as *pure half-integer*, and let us notate as $\mathfrak{H}_2^n \coloneqq \mathfrak{X}_2^n \cap \{0, 1/2\}^{|E^n|}$ the set of such points. Consequence of this last result is first of all the fact that $\{\mathrm{Gap}_n\}_n$ is a non-decreasing sequence, and then that in order to improve estimates of Gap_n starting from Gap_{n-1} it is enough to consider points in $\mathfrak{H}_2^n$, bypassing the other elements of $\mathfrak{X}_2^n$.

2. Combinatorics of Half-Integer Solutions

Having delimited the subject of this work to the search for high-gap half-integer extreme points of the ASEP polytope $\mathfrak{P}^n$, we now present a theoretical framework and new method derived from it to procedurally generate such instances. The main idea is to split the travelling salesman in two halves, and ask for each cloven salesman to be travelling along a vertex-disjoint cycle cover in place of a tour. This way an instance of $\mathfrak{X}_2^n$ is seen as the sum of vectors associated with two vertex-disjoint cycle covers, effectively reducing the problem of generating half-integer points to the easier problem of generating the two covers and checking whether their sum satisfies some wanted requirements.

2.1 Structure Theorem of $\mathfrak{X}_2^n$

Let us first focus on the vectors associated with vertex-disjoint cycle covers in $\mathcal{K}_n$, and let us call $\mathfrak{C}^n \subset \{0,1\}^{|E^n|}$ the set of these vectors. The space of these vectors is in fact characterized by the same constraints of the ATSP (1.1) without the subtour elimination constraints (1.4),

$$\mathfrak{C}^n = \left\{ \overline{\boldsymbol{y}} \in \mathbb{R}^{|E^n|} \;\middle|\; (1.2), (1.3), (1.5), (1.6) \right\}.$$

By additionally dropping the integrality constraints (1.6), the set of extreme points of the resulting polytope $\mathfrak{Q}^n$ is equivalent to $\mathfrak{C}^n$, since Birkhoff showed in [3] how such extreme points are always integral. This in turn implies that a search algorithm for the vectors $\overline{\boldsymbol{y}} \in \mathfrak{C}^n$ can be achieved by means of a linear program, making the problem P time complex; this strategy however is not employed in the present work.

We start by showing that it is possible to represent any half-integer solution as sum of two cycle covers: the claim is that for every $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$ there exists two cycle covers $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$ such that $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2$. In order to prove it, we first need to introduce an auxiliary lemma.

Lemma 2.1. *Let $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$ and let $\mathcal{X}_{\overline{\boldsymbol{x}}}^n$ be its associated graph. Let C^1 be a cycle cover of $\mathcal{X}_{\overline{\boldsymbol{x}}}^n$ with characteristic vector $\overline{\boldsymbol{y}}^1 \in \mathfrak{C}^n$ and let $\overline{\boldsymbol{y}}^2 = 2\overline{\boldsymbol{x}} - \overline{\boldsymbol{y}}^1$. Then $\overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$.*

Proof. C^1 being a cover of $\mathcal{X}_{\overline{\boldsymbol{x}}}^n$ implies that $\operatorname{supp}_{E^n}\left(\overline{\boldsymbol{y}}^1\right) \subseteq \operatorname{supp}_{E^n}\left(\overline{\boldsymbol{x}}\right)$, which in turn means that $\mathbf{0} \leq \overline{\boldsymbol{y}}^1 \leq 2\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{y}}^2 = 2\overline{\boldsymbol{x}} - \overline{\boldsymbol{y}}^1 \geq \mathbf{0}$. The boundary constraints (1.5) are thus satisfied by $\overline{\boldsymbol{y}}^2$, which also satisfies the out-degree constraints (1.2) since $\forall\, w \in V^n$

$$\sum_{uv \in \delta^+(w)} 4\overline{y}_{uv}^2 = \sum_{uv \in \delta^+(w)} 4\left(2\overline{x}_{uv} - \overline{y}_{uv}^1\right) = 2\sum_{uv \in \delta^+(w)} 4\overline{x}_{uv} - \sum_{uv \in \delta^+(w)} 4\overline{y}_{uv}^1 = 2 \cdot 1 - 1 = 1.$$

Analogously for the in-degree constraints (1.3). The integrality constraints (1.6) are trivially satisfied since both $2\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{y}}^1$ are integral. □

We are now ready to prove the main result of this work, fundamental to our construction, that serves as a structure theorem for half-integer extreme points.

Theorem 2.2 (structure theorem of $\mathfrak{X}_2^n$)**.** *For each $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$, there exist $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$ such that $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2$.*

Proof. Thanks to Lemma 2.1 it suffices to show that a cycle cover of $\mathcal{X}_{\overline{\boldsymbol{x}}}^n$ exists. This is equivalent to the existence of a perfect matching on the bipartite graph

$$\widetilde{\mathcal{X}} \coloneqq \left(V^n \times \{0,1\}, \widetilde{E}_{\overline{\boldsymbol{x}}}^n\right), \qquad \text{with} \quad \widetilde{E}_{\overline{\boldsymbol{x}}}^n \coloneqq \{(u,0)(v,1) \mid uv \in \operatorname{supp}_{E^n}(\overline{\boldsymbol{x}})\}.$$

The existence of such matching is granted by Hall's marriage theorem, whose hypotheses require us to prove that $\left|\overline{S}\right| \leq \left|N^+_{\mathcal{X}_{\overline{\boldsymbol{x}}}^n}\left(\overline{S}\right)\right|$ for any $\overline{S} \in \mathscr{P}(V^n \times \{0\}) \cup \mathscr{P}(V^n \times \{1\})$. Let $\overline{S} \coloneqq S \times \{0\}$ with $S \subseteq V^n$ and $N_{\overline{S}} \coloneqq N^+_{\mathcal{X}_{\overline{\boldsymbol{x}}}^n}\left(\overline{S}\right)$; we see that

$$\bigcup_{u \in \overline{S}} \left\{uv \in \widetilde{E}_{\overline{\boldsymbol{x}}}^n \;\middle|\; v \in \delta^+(u)\right\} \subseteq \bigcup_{v' \in N_{\overline{S}}} \left\{u'v' \in \widetilde{E}_{\overline{\boldsymbol{x}}}^n \;\middle|\; u' \in \delta^-(v')\right\},$$

from which it follows

$$\left|\overline{S}\right| = \sum_{u \in \overline{S}} 1 = \sum_{u \in \overline{S}} \Big(\sum_{v \in \delta^+(u)} 4\overline{x}_{uv} \Big) \leq \sum_{v' \in N_{\overline{S}}} \Big(\sum_{u' \in \delta^-(v')} 4\overline{x}_{u'v'} \Big) = \sum_{v' \in N_{\overline{S}}} 1 = \left|N_{\overline{S}}\right|,$$

that is $\left|\overline{S}\right| \leq \left|N^+\left(\overline{S}\right)\right|$ (analogous for $\overline{S} = S \times \{1\}$). Thus the hypotheses of Hall's marriage theorem hold, from which the thesis follows. □

Remark 2.3. Both Lemma 2.1 and Theorem 2.2 do not take advantage of the fact that $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$ satisfies the subtour elimination constraints (1.4), the results therefore also hold for any $\overline{\boldsymbol{x}} \in \mathfrak{Q}^n \cap \{0, 1/2, 1\}^{|E^n|}$, that is the half-integer solutions of the remaining constraints (1.2), (1.3) and (1.5). This also means that the results of Theorem 2.2 can be extended via induction to $\mathfrak{X}_\alpha^n$ for $\alpha \geq 3$, proving that the elements $\overline{\boldsymbol{x}} \in \mathfrak{X}_\alpha^n$ are expressible as $\overline{\boldsymbol{x}} = \frac{1}{\alpha}\sum_{i=1}^{\alpha} \overline{\boldsymbol{y}}^i$ with $\overline{\boldsymbol{y}}^i \in \mathfrak{C}^n$, $i \in \{1, ..., \alpha\}$. In fact, similarly to Lemma 2.1 we see that if a cycle cover $\overline{\boldsymbol{y}}^\alpha \in \mathfrak{C}^n$ of $\mathcal{X}_{\overline{\boldsymbol{x}}}^n$ exists then $\overline{\boldsymbol{x}}' \coloneqq \frac{1}{\alpha-1}(\alpha\overline{\boldsymbol{x}} - \overline{\boldsymbol{y}}^\alpha)$ belongs to $\mathfrak{Q}^n \cap \left\{0, \frac{1}{\alpha-1}, \ldots, 1\right\}^{|E^n|}$ (but not to $\mathfrak{X}_{\alpha-1}^n$ in general, as $\overline{\boldsymbol{x}}'$ may not satisfy the subtour elimination constraints). Hence from the inductive hypothesis we would have $\overline{\boldsymbol{x}}' = \frac{1}{\alpha-1}\sum_{i=1}^{\alpha-1} \overline{\boldsymbol{y}}^i$ with $\overline{\boldsymbol{y}}^i \in \mathfrak{C}^n$ for $i \in \{1, \ldots, \alpha-1\}$ and therefore

$$\overline{\boldsymbol{x}} = \frac{\alpha-1}{\alpha}\overline{\boldsymbol{x}} + \frac{1}{\alpha}\overline{\boldsymbol{y}}^\alpha = \frac{\cancel{\alpha-1}}{\alpha}\frac{1}{\cancel{\alpha-1}}\sum\nolimits_{i=1}^{\alpha-1} \overline{\boldsymbol{y}}^i + \frac{1}{\alpha}\overline{\boldsymbol{y}}^\alpha = \sum\nolimits_{i=1}^{\alpha} \overline{\boldsymbol{y}}^i.$$

This structure theorem entails that any enumeration of half-integer solutions in $\mathfrak{X}_2^n$ as sum of cycle covers is exhaustive. Asking for the converse of this result naturally follows: for any choice of $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$ does it hold that $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2$ is in $\mathfrak{X}_2^n$? We see that this is not guaranteed in the general case, despite $\overline{\boldsymbol{x}}$ trivially being in $\{0, 1/2, 1\}^{|E^n|}$. There are in fact two possible cases that lead to $\overline{\boldsymbol{x}} \notin \mathfrak{X}_2^n$, that are

- ▷ $\overline{\boldsymbol{x}} \notin \mathfrak{P}^n$, because it might violate the subtour elimination constraints (1.4) (a trivial example of this can be constructed by taking $\overline{\boldsymbol{y}}^1 \in \mathfrak{C}^n \setminus \mathfrak{P}^n$ and $\overline{\boldsymbol{y}}^2 = \overline{\boldsymbol{y}}^1$, for which $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2 = \overline{\boldsymbol{y}}^1 \notin \mathfrak{P}^n$);
- ▷ $\overline{\boldsymbol{x}} \in \mathfrak{P}^n$ yet $\overline{\boldsymbol{x}} \notin \mathfrak{X}^n$, because it is not granted that $\overline{\boldsymbol{x}}$ is an extreme point of $\mathfrak{P}^n$ (Figure 2.1 shows an example of this case).

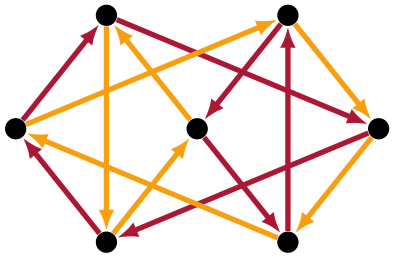


Figure 2.1: A solution $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2$ that is not an extreme point of $\mathfrak{P}^n$, as for all $S \in \mathscr{S}^n$ we have $\sum_{uv \in \delta^+(S)} x_{uv} > 1$ and thus there are no tight subtour elimination constraints.

The strongest results on the properties of $\overline{\boldsymbol{x}}$ are instead the following.

Proposition 2.4. *Let $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$ and let $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2$. Then $\overline{\boldsymbol{x}}$ is made out of $\{0, 1/2, 1\}$-components and satisfies the out-degree (1.2), in-degree (1.3) and bound constraints (1.5). Equivalently $\overline{\boldsymbol{x}} \in \mathfrak{Q}^n \cap \{0, 1/2, 1\}^{|E^n|}$.*

Proof. $\overline{\boldsymbol{x}}$ is trivially composed of only $0, 1/2, 1$. Satisfaction of constraints (1.2), (1.3) and (1.5) comes from the weighted sum of the same constraints that $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$ satisfy. □

Corollary 2.5. *Let $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2 \in \mathfrak{C}^n$ and let $\overline{\boldsymbol{x}} = \frac{1}{2}\overline{\boldsymbol{y}}^1 + \frac{1}{2}\overline{\boldsymbol{y}}^2$. Then $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$ if and only if $\overline{\boldsymbol{x}}$ satisfies the subtour elimination constraints (1.4) and is an extreme point of $\mathfrak{P}^n$.*

2.2 Encoding

We now focus on the underlying cycle covers to establish a framework for their efficient generation. To begin, let us introduce a method for encoding each cycle cover, which allows us to reduce the problem of generating non-isomorphic cycle covers to that of generating their encodings under certain clauses.

Cover Encodings

Definition 2.6 (cover encoding)**.** Let $n, N \in \mathbb{N}, n \geq 2, 1 \leq N \leq n$ and let $p_1, ..., p_N$ be a integer partition of n, that is $p_1, ..., p_N \in \mathbb{N}^{>0}$ such that $p_1 + ... + p_N = n$ with $p_1 \geq ... \geq p_N \geq 2$. We define

$$\xi = [\, k_1^1 \dots k_{p_1}^1 \mid k_1^2 \dots k_{p_2}^2 \mid \cdots \mid k_1^N \dots k_{p_N}^N \,],$$

with $k_i^j \in V^n$ and $k_i^j \neq k_{i'}^{j'}$ for all $j, j', i, i' \in \mathbb{N}^{>0},\ j, j' \leq N,\ i \leq p_j,\ i' \leq p_{j'}$ as the encoding of the cover C of $\mathcal{K}_n$ defined by

$$C \coloneqq \bigcup_{j=1}^{N} \left(\left\{ k_i^j k_{i+1}^j \;\middle|\; i \in \mathbb{N}^{>0}, i \leq p_j - 1 \right\} \cup \left\{ k_{p_j}^j k_1^j \right\} \right).$$

Let $(p_1, ..., p_N)$ be known as the *partition* of the encoding ξ. We also ask for the following two conditions to be met by the encoding:

1. $k_1^j < k_i^j$ for all $1 \leq j \leq N,\ 2 \leq i \leq p_j$;
2. $k_1^j < k_1^{j'}$ for any $j < j'$ such that $p_j = p_{j'}$.

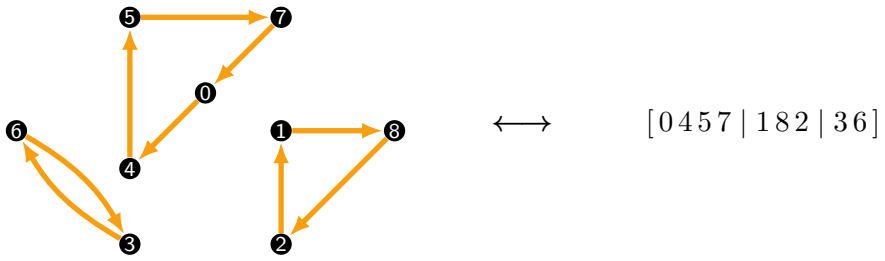


Figure 2.2: Example cover with its associated encoding.

This last requirements are needed to pinpoint a specific encoding among all those that return the same cover C. It also can be seen how any cover C of a graph on V^n has an associated encoding, which can be obtained by following Algorithm 2.1: the selection of the cycles of C in an ordinate manner ensures that the encoding requirements (1.) and (2.) are met. The existence of the algorithm also implies that for a fixed vertex set V^n the encoding is unique.

Algorithm 2.1 Cover encoding

▷ **Input** A cycle cover C

▷ **Output** A cover encoding $[k_1^1 \dots k_{p_1}^1 \mid k_1^2 \dots k_{p_2}^2 \mid \dots \mid k_1^N \dots k_{p_N}^N]$

1. Sort the cycles in C by decreasing cardinality
2. Sort the same-cardinality cycles in C by increasing minimal vertex
3. $j \leftarrow 1$
4. **for** $c \in$ (sorted set of cycles of C) **do**
5. $\quad p_j \leftarrow |c|$
6. $\quad k_1^j \leftarrow$ minimal vertex in c
7. $\quad$ **for** $i \in \{2, \dots, p_j\}$ **do**
8. $\quad\quad$ Find $uv \in \delta^+(k_{i-1}^j) \cap C$
9. $\quad\quad k_i^j \leftarrow v$
10. $\quad$ **end for**
11. $\quad j \leftarrow j + 1$
12. **end for**

Given a permutation $\sigma\colon \{0, \dots, n-1\} \hookrightarrow \{0, \dots, n-1\}$ and an encoding ξ, we write $\sigma(\xi)$ to identify the *translation* of the encoding via σ, that is the element-wise application of σ to ξ together with an eventual reordering of its elements so that the resulting encoding satisfies both requirements (1.) and (2.) of Definition 2.6. The cover encoded by the translation $\sigma(\xi)$ is trivially isomorphic to that encoded by ξ, as the permutation σ itself is an arc-preserving isomorphism between the two.

Remark 2.7. The mentioned reordering is equivalent to and can be implemented as the execution of Algorithm 2.1 to the result of the application of σ to ξ.

If we relax the requisites of definition Definition 2.6 by letting the partition of the encoding be the integer partition of a $m \leq n$, we are able to define the concatenation $\xi_1 + \xi_2$ of two such encoding-slices ξ_1 and ξ_2, with $\sum_{j=1}^{M'} p_j = m', \sum_{j=1}^{M''} q_j = m''$ as

partitions respectively and $m' + m'' \leq n$, as

$$\xi_1 = [k_1^1 \dots k_{p_1}^1 \mid \cdots \mid k_1^{M'} \dots k_{p_{M'}}^{M'}], \qquad \xi_2 = [h_1^1 \dots h_{q_1}^1 \mid \cdots \mid h_1^{M''} \dots h_{q_{M''}}^{M''}],$$
$$\xi_1 + \xi_2 \coloneqq [k_1^1 \dots k_{p_1}^1 \mid \cdots \mid k_1^{M'} \dots k_{p_{M'}}^{M'} \mid h_1^1 \dots h_{q_1}^1 \mid \cdots \mid h_1^{N''} \dots h_{q_{M''}}^{M''}],$$

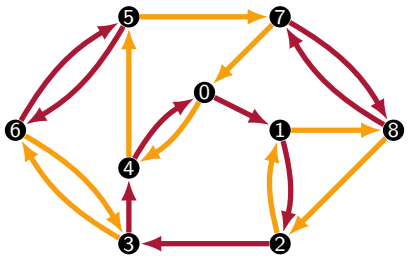
always ensuring that the resulting encoding-slice satisfies the other requirements of a cover encoding (e.g. ξ_1 and ξ_2 must have no common elements).

Cover-set Encodings

We now go on to consider sets of cycle covers, again defining a system to encode them.

Definition 2.8 (cover-set encoding)**.** Let $\mathscr{C}$ be a set of cycle covers of a $\mathcal{K}_n$. We encode $\mathscr{C}$ as the ordered set of encodings of its covers, where the ordering is obtained by the following two criteria:

1. descending lexicographic order of the encodings' partitions;
2. for identical partitions, ascending lexicographic order of the encodings themselves.



$\longleftrightarrow$

$[0\,1\,2\,3\,4 \mid 5\,6 \mid 7\,8]$
$[0\,4\,5\,7 \mid 1\,8\,2 \mid 3\,6]$

Figure 2.3: Example cover-set with its associated encoding (in standard form).

Standard form Although the cover-set encoding is also unique when the vertex set V^n is fixed (a direct consequence of the uniqueness of the cover encodings), this is not true in general when allowing for the vertices to be relabeled, a procedure that trivially preserves isomorphism. It would therefore be desirable to take advantage of vertex relabeling to reduce the number of isomorphic instances in our generating procedure. Our aim is to find a permutation of the labels transforming the first of the cover encodings to the form

$$\left[\, 0 \dots (p_1 - 1) \;\middle|\; p_1 \dots (p_1 + p_2 - 1) \;\middle|\; \cdots \;\middle|\; (n - p_N - 1) \dots (n - 1) \,\right]. \tag{2.1}$$

We will refer to a cover-set encoding in this configuration as a *standard* encoding or an encoding in *standard form*. Algorithm 2.2 shows a procedure to retrieve the standard form of any cover-set encoding, and the existence of a standard form isomorphic to any given encoding also proves that by generating all possible standard encodings, all cover-sets up to isomorphism are likewise generated.

Canonic form The standard form of a cover-set encoding however is not unique, as in general the result of Algorithm 2.2 depends on the random selections made during its computation. We define as *canonic form* of a cover-set encoding the standard form that achieves the minimum in a lexicographic order among all possible standard forms (the

Algorithm 2.2 Standard cover-set encoding

▷ **Input** A cover-set encoding $\Xi = (\xi_1, \ldots, \xi_M)$, $(p_1, ..., p_N)$ partition of ξ_1

▷ **Output** A cover-set encoding Ξ' in standard form

1. Let $\sigma\colon \{0, \ldots, n-1\} \hookrightarrow \{0, \ldots, n-1\}$ be the translating permutation
2. Select $\xi_m \in \Xi$ with the same partition $(p_1, ..., p_N)$ as ξ_1
3. Let $\xi_m = [k_1^1 \ldots k_{p_1}^1 \mid \cdots \mid k_1^N \ldots k_{p_N}^N]$
4. $P \leftarrow \emptyset$; $S \leftarrow 0$
5. **for** $j' \in \{1, \ldots, N\}$ **do**
6. Select j such that $p_j = p_{j'}$ and $j \notin P$
7. $P \leftarrow P \cup \{j\}$
8. Select $s \in \{0, \ldots, p_j - 1\}$
9. **for** $i \in \{1, \ldots, p_j\}$ **do**
10. $t \leftarrow s + i \mod p_j$
11. $\sigma(k_i^j) \leftarrow S + t$
12. **end for**
13. $S \leftarrow S + p_j$
14. **end for**
15. $\Xi' \leftarrow (\sigma(\xi_1), \ldots, \sigma(\xi_M))$
16. Reorder Ξ' following criteria (1.) and (2.) of Definition 2.8

comparison between elements of the standard forms is again done lexicographically). By construction the canonic form is therefore unique for each set-cover encoding.

While, analogously to standard forms, generating all canonic forms would suffice to generate all sets of covers up to isomorphism, this task turns out to be much harder. Differently from standard encodings — for which as we will see in Section §2.4 a direct generation is possible — producing canonic instances indeed requires exploring all its possible standard translations to check whether the minimum has been reached, obviously an extremely demanding operation. Notice also how it is in fact equivalent to generating all standard forms and then checking for their canonicity afterwards.

Additionally, counterexamples as the one shown in Figure 2.4 display how two isomorphic cover-sets may have different canonic forms (sometimes even with different partitions): the canonic form is therefore not strong enough to identify the isomorphism class of a cover-set.

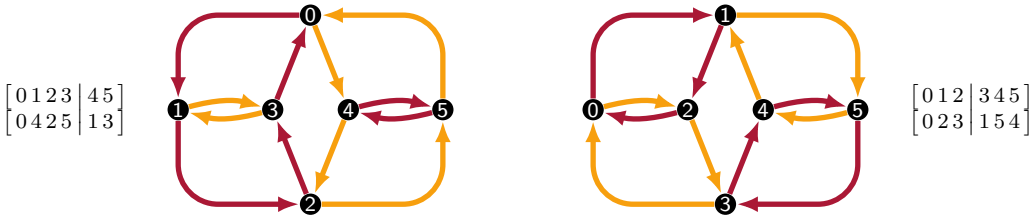


Figure 2.4: A cover-set with two different canonic encodings.

It is therefore equivalent and better for us to generate all standard encodings, since it leaves us the choice of whether to check afterwards for their canonicity or directly for their isomorphism class with faster algorithms.

2.3 Property Checking

At the end of Section §2.1 we showed how by taking any two covers we are not guaranteed that their combination falls in $\mathfrak{P}^n$ or $\mathfrak{X}^n$, while in Section §2.2 we showed how by taking any standard form encoding of a cover-set there is no guarantee that it is uniquely associated with an isomorphism class or even that it is a canonic form. In this section we introduce procedures to check for those properties on a general set of covers $\{\overline{\boldsymbol{y}}^1, \dots, \overline{\boldsymbol{y}}^\alpha\} \subset \mathfrak{C}^n$ and on the combination $\overline{\boldsymbol{x}} = \frac{1}{\alpha}\sum_{i=1}^{\alpha} \overline{\boldsymbol{y}}^i$ in $\mathfrak{X}^n_\alpha$.

2.3.1 Canonicity

It was remarked how checking for the canonicity of the cover-set encoding requires looking at all different standard translations of the encoding to find whether the given one is minimal. Despite the ability to implement an early stopping of the procedure as soon as a lexicographically lower standard translation has been found, greatly reducing the amount of translations that need to be explored, the computational cost of the method remains very high. Adding to this the fact that checking for the isomorphism class is in any case required, as canonicity alone cannot detect isomorphism, skipping the canonicity check may be the most efficient approach, a result to which empirical tests also point as will be shown in Section §3.1.

2.3.2 Isomorphism Class

In order to check for the isomorphism class of an instance in our implementation we make use of the NAUTY software package [23], which provides state-of-the-art routines for computing graph isomorphisms. Among the provided routines, NAUTY allows for the powerful computation of a binary certificate that uniquely identifies the class of isomorphism of a graph. Unfortunately, NAUTY does not support the specification of arc weights, only providing support for vertex colorings with colors that are to be preserved by isomorphisms. We are however able to reduce the problem of finding arc-weight-preserving graph isomorphisms to the problem of finding vertex-coloring-preserving graph

Algorithm 2.3 Vertex-colored graph

▷ **Input** A weighted graph $\mathcal{X}^n_{\overline{\boldsymbol{x}}} = (V^n, \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}}))$ associated to $\overline{\boldsymbol{x}} \in \mathfrak{X}^n_\alpha$

▷ **Output** A new graph $\mathcal{X}' = (V', E')$ with a vertex-coloring partition $\mathscr{C} = \{\mathscr{C}_1, ..., \mathscr{C}_\alpha\}$

1. $V' \leftarrow V^n$; $E' \leftarrow \emptyset$; $n' \leftarrow n$
2. $\{\mathscr{C}_1, ..., \mathscr{C}_\alpha\} \leftarrow \{V^n, \emptyset, \dots, \emptyset\}$
3. **for** $uv \in \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})$ **do**
4. $\quad k \leftarrow \alpha \cdot \overline{x}_{uv}$
5. $\quad$ **if** $k = 1$ **then**
6. $\quad\quad E' \leftarrow E' \cup \{uv\}$
7. $\quad$ **else**
8. $\quad\quad V' \leftarrow V' \cup \{n'\}$; $\mathscr{C}_k \leftarrow \mathscr{C}_k \cup \{n'\}$
9. $\quad\quad E' \leftarrow E' \cup \{un', n'v\}$
10. $\quad\quad n' \leftarrow n' + 1$
11. $\quad$ **end if**
12. **end for**

isomorphisms thanks to Algorithm 2.3, which produces a vertex-colored graph associated with an arc-weighted one.

2.3.3 Subtour Elimination and Extremality

Checking for the fulfillment of the subtour elimination constraint can be achieved by directly verifying the satisfaction by $\overline{\boldsymbol{x}}$ of the subtour elimination inequalities (1.4). Having thus verified that $\overline{\boldsymbol{x}}$ is a feasible solution ($\overline{\boldsymbol{x}} \in \mathfrak{P}^n$), we are now left with the task of checking whether $\overline{\boldsymbol{x}}$ is also an extreme point of $\mathfrak{P}^n$ ($\overline{\boldsymbol{x}} \in \mathfrak{X}^n$). Linear programming theory shows how this is equivalent to asking for $\overline{\boldsymbol{x}}$ to be a basic solution, that is a solution for which the number of linearly independent active constraints (constraints that the solution satisfies with equality) is equal to the dimension of the solution space. In other words, let the set $A = \left\{\boldsymbol{a}^1, \boldsymbol{a}^2, ..., \boldsymbol{a}^T\right\}$ contain the vectors of coefficients of active ASEP constraints ($\boldsymbol{a}^i \cdot \overline{\boldsymbol{x}} = c^i$); then $\overline{\boldsymbol{x}}$ is extreme if and only if $\operatorname{rank}(A) = n^2 - n = |E^n|$. Therefore the extremality of $\overline{\boldsymbol{x}}$ can be verified by finding all its active constraints[1] and checking whether they exhibit a maximal rank. Computing the rank of a matrix though is a computationally expensive task, especially when dealing with large vector spaces. Our aim is therefore to elaborate a faster method to evaluate the rank-maximality of the active constraints, by leveraging the structure of the problem and by tailoring it to the particular case of $\alpha = 2$ (that is $\overline{\boldsymbol{x}} \in \mathfrak{P}_2^n$), which constitutes the focus of our computational efforts.

Let us first analyze the coefficient vectors of the ASEP constraints. We see that they are all binary vectors in the space $\{0,1\}^{|E^n|}$, and they are fixed for any choice of n. Let us refer as $\deg_u^{+,n}, \deg_v^{-,n}$ to the vectors relative to respectively the out-degree and in-degree constraints of $u, v \in V^n$, as $\operatorname{bnd}_{uv}^n$ to the vector of the bound constraint relative to $uv \in E^n$, and as sep_S^n to the vector of the subtour elimination constraint of $S \in \mathscr{S}$.

	Constraint	Index	Vector	Number	# Active
(1.2)	Out-Degree	$u \in V^n$	$\deg_u^{+,n}$	n	n
(1.3)	In-Degree	$v \in V^n$	$\deg_v^{-,n}$	n	n
(1.4)	Subtour Elimination	$S \in \mathscr{S}^n$	sep_S^n	$2^n - 2n - 2$	
(1.5)	Bound	$uv \in E^n$	$\operatorname{bnd}_{uv}^n$	$n^2 - n$	$\geq n^2 - 3n$

Table 2.1: Summary of ASEP constraints.

It is convenient to represent them not as simple horizontal or vertical vectors, but as $n \times n$ matrices without the main diagonal and with the uv-entry of the vector as the (u, v)-cell. The $\deg_u^{+,n}$, $\deg_u^{-,n}$ and $\operatorname{bnd}_{uv}^n$ thus become "row", "column" and "one-hot" matrices, that is

[1] This can be done simultaneously with the subtour elimination check by keeping track of which subtour elimination are satisfied with equality; to these the active boundary constraints as well as the always-active degree constraint are then added.

$$\deg_u^{+,n} = \begin{array}{c} \\ u \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} \searrow & 0 & 0 & 0 & \cdot\cdot & 0 \\ 0 & \searrow & 0 & 0 & & 0 \\ \vdots & & \searrow & \vdots & & \vdots \\ \boxed{1} & \boxed{1} & \boxed{1} & \searrow & \cdot\cdot & 1 \\ \vdots & & & \vdots & \searrow & \vdots \\ 0 & 0 & 0 & 0 & \cdot\cdot & \searrow \end{pmatrix}, \qquad \deg_v^{-,n} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} & & & v & & \\ \searrow & 0 & \cdot\cdot & \boxed{1} & \cdot\cdot & 0 \\ 0 & \searrow & & 1 & & 0 \\ 0 & 0 & \searrow & 1 & & 0 \\ 0 & 0 & \cdot\cdot & \searrow & \cdot\cdot & 0 \\ \vdots & & & \vdots & \searrow & \vdots \\ 0 & 0 & \cdot\cdot & 1 & \cdot\cdot & \searrow \end{pmatrix},$$

$$\mathrm{bnd}_{uv}^n = \begin{array}{c} \\ u \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} & & v & & \\ \searrow & 0 & 0 & \cdot\cdot & 0 \\ 0 & \searrow & \boxed{1} & & 0 \\ 0 & 0 & \searrow & \cdot\cdot & 0 \\ \vdots & & \vdots & \searrow & \vdots \\ 0 & 0 & 0 & \cdot\cdot & \searrow \end{pmatrix}.$$

With regards to the constraints activation, we observe that the $2n$ degree constraints are always active since they are equalities, and are all but one linearly independent. Bound constrains instead are all linearly independent and are active for those arcs $uv \in E^n$ that have $\overline{x}_{uv} = 0$, thus the corresponding set of active vectors has at least $n^2 - 3n$ items (obtained by keeping in mind that $\overline{\boldsymbol{x}} \in \mathfrak{P}_2^n$) and corresponds to

$$\{\mathrm{bnd}_{uv}^n \mid uv \in E^n \setminus \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})\}.$$

Let us furthermore notate $E_{\mathtt{ACT}}^n(\overline{\boldsymbol{x}}) \coloneqq E^n \setminus \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})$ and the set of $S \in \mathscr{S}^n$ such that sep_S^n is the vector associated with an active constraint as $\mathscr{S}_{\mathtt{ACT}}^n(\overline{\boldsymbol{x}})$. Finally let us define

$$\mathtt{DEG}_V^{+,n} \coloneqq \mathrm{span}\left(\left\{\deg_u^{+,n} \;\middle|\; u \in V\right\}\right), \qquad \mathtt{SEP}_{\mathscr{S}}^n \coloneqq \mathrm{span}\left(\left\{\mathrm{sep}_S^n \;\middle|\; S \in \mathscr{S}\right\}\right),$$
$$\mathtt{DEG}_V^{-,n} \coloneqq \mathrm{span}\left(\left\{\deg_v^{-,n} \;\middle|\; v \in V\right\}\right), \qquad \mathtt{BND}_E^n \coloneqq \mathrm{span}\left(\left\{\mathrm{bnd}_{uv}^n \;\middle|\; uv \in E\right\}\right),$$

as well as $\mathtt{ACT}_{\overline{\boldsymbol{x}}}^n \coloneqq \mathtt{DEG}_{V^n}^n + \mathtt{SEP}_{\mathscr{S}_{\mathtt{ACT}}^n(\overline{\boldsymbol{x}})}^n + \mathtt{BND}_{E_{\mathtt{ACT}}^n(\overline{\boldsymbol{x}})}^n$ and $\mathtt{DEG}_V^n \coloneqq \mathtt{DEG}_V^{+,n} + \mathtt{DEG}_V^{-,n}$.

The vectors bnd_{uv}^n are fundamentally the standard basis vectors: $\mathtt{BND}_E^n$ is thus the subspace of the vectors of $\mathbb{R}^{|E^n|}$ that have non-null components indexed by E, which we will notate also as $\langle E \rangle^n \coloneqq \mathtt{BND}_E^n$. Our objective is thus to check whether

$$\mathtt{ACT}_{\overline{\boldsymbol{x}}}^n \stackrel{?}{=} \langle E^n \rangle^n = \mathbb{R}^{|E^n|}. \tag{2.2}$$

To achieve this, the remainder of the section is devoted to designing an algorithm relying on a partitioning of the non-zero arcs of $\overline{\boldsymbol{x}}$ based on the degree constraints vectors, and its successive coarsening by the addition of the vectors associated with active subtour elimination constraints. The result of this will be the circuit extremality algorithm defined in Proposition 2.19 and Algorithm 2.4.

1-Arcs Reduction

First we observe that whenever the vector $\overline{\boldsymbol{x}}$ has a component $\overline{x}_{\overline{u}\,\overline{v}} = 1$, we are able to reduce the problem to a lower dimensional one. This comes as a consequence of the following proposition, where $\mathrm{pr}_W(\cdot)$ is intended as the projection on the subspace W.

Proposition 2.9. *Let $\overline{u}\,\overline{v} \in E^n$ such that $\overline{x}_{\overline{u}\,\overline{v}} = 1$, let $E_{\overline{u}\,\overline{v}} \coloneqq \delta^+(\overline{u}) \cup \delta^-(\overline{v}) \cup \{\overline{v}\,\overline{u}\}$ and $E^{\complement}_{\overline{u}\,\overline{v}} \coloneqq E^n \setminus E_{\overline{u}\,\overline{v}}$. Then the condition $\mathtt{ACT}^n_{\overline{\boldsymbol{x}}} = \mathbb{R}^{|E^n|}$ of (2.2) is equivalent to verifying*

$$\mathrm{pr}_{\langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n}\left(\mathtt{ACT}^n_{\overline{\boldsymbol{x}}}\right) = \langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n .$$

Proof. The degree and bound constraints imply $\left(\delta^+(\overline{u}) \cup \delta^-(\overline{v})\right) \setminus \{\overline{u}\,\overline{v}\} \subseteq E^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})$ since

$$\overline{x}_{uv} = 0 \quad \forall\, uv \in \left(\delta^+(\overline{u}) \cup \delta^-(\overline{v})\right) \setminus \{\overline{u}\,\overline{v}\} .$$

Also it must be $\overline{x}_{\overline{v}\,\overline{u}} = 0$ (and so $\overline{v}\,\overline{u} \in E^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})$) as by subtour elimination for the set $S = \{\overline{u}, \overline{v}\}$, for which it holds $\delta^+(S) = \delta^+(\overline{u}) \cup \delta^+(\overline{v}) \setminus \{\overline{u}\,\overline{v}, \overline{v}\,\overline{u}\}$, we have

$$1 \leq \sum_{uv \in \delta^+(S)} 4\overline{x}_{uv} = \sum_{uv \in \delta^+(\overline{u})} 4\overline{x}_{uv} + \sum_{uv \in \delta^+(\overline{v})} 4\overline{x}_{uv} - \overline{x}_{\overline{u}\,\overline{v}} - \overline{x}_{\overline{v}\,\overline{u}} = 1 + 1 - 1 - \overline{x}_{\overline{v}\,\overline{u}} = 1 - \overline{x}_{\overline{v}\,\overline{u}}.$$

Finally, we can see that $\mathtt{BND}^n_{E_{\overline{u}\,\overline{v}}} \subseteq \mathtt{ACT}^n_{\overline{\boldsymbol{x}}}$, as also $\mathrm{bnd}^n_{\overline{u}\,\overline{v}}$ is writable as a linear combination of elements of $\mathtt{ACT}^n_{\overline{\boldsymbol{x}}}$,

$$\mathrm{bnd}^n_{\overline{u}\,\overline{v}} = \mathrm{deg}^{+,n}_{\overline{u}} - \sum\nolimits_{\overline{u}v \in \delta^+(\overline{u})} \mathrm{bnd}^n_{\overline{u}v} = \mathrm{deg}^{-,n}_{\overline{v}} - \sum\nolimits_{u\overline{v} \in \delta^-(\overline{v})} \mathrm{bnd}^n_{u\overline{v}}.$$

The thesis comes from the decomposition $\mathbb{R}^{|E^n|} = \langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n \oplus \langle E_{\overline{u}\,\overline{v}}\rangle^n$. □

Thus it suffices to check for the rank maximality of $\mathtt{ACT}^n_{\overline{\boldsymbol{x}}}$ on the subspace $\langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n$, orthogonal to $\langle E_{\overline{u}\,\overline{v}}\rangle^n$. Notice how the projection on this subspace of any vector $\boldsymbol{a} \in \mathbb{R}^{|E^n|}$ is equivalent to setting all components indexed by elements of $E_{\overline{u}\,\overline{v}}$ to 0,

$$\left(\mathrm{pr}_{\langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n}(\boldsymbol{a})\right)_{uv} = \begin{cases} a_{uv} & \text{if } uv \notin E_{\overline{u}\,\overline{v}}, \\ 0 & \text{if } uv \in E_{\overline{u}\,\overline{v}}. \end{cases}$$

Let us now take into consideration a new vector $\overline{\boldsymbol{x}}' \in \mathfrak{P}_2^{n-1}$, obtained by identifying the two vertices $\overline{u}$ and $\overline{v}$ to a single new vertex $\overline{w}$ and defined similarly to equation (1.9)

$$\overline{x}'_{ab} = \overline{x}_{\rho(ab)}, \qquad \text{with } \rho(ab) \coloneqq \begin{cases} \overline{v}b & \text{if } a = \overline{w}, \\ a\overline{u} & \text{if } b = \overline{w}, \\ ab & \text{otherwise}, \end{cases}$$

and accordingly $V^{n-1} = \{\overline{w}\} \cup V^n \setminus \{\overline{u}, \overline{v}\}$. Let us furthermore introduce the linear application

$$[\![\cdot]\!] \colon \langle E^n\rangle^n \to \left\langle E^{n-1}\right\rangle^{n-1} = \mathbb{R}^{|E^{n-1}|}, \qquad \left([\![\boldsymbol{a}]\!]\right)_{ab} = a_{\rho(ab)}.$$

Essentially $\overline{\boldsymbol{x}}' = [\![\overline{\boldsymbol{x}}]\!]$, and since the function ρ is such that $\rho\left(E^{n-1}\right) = E^{\complement}_{\overline{u}\,\overline{v}}$, we have that

$$\left[\!\left[\mathrm{pr}_{\langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n}(\boldsymbol{a})\right]\!\right] = [\![\boldsymbol{a}]\!] . \tag{2.3}$$

It is also immediate to see that when restricted to $\langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n$ in its domain $[\![\cdot]\!]$ is an isomorphism, as $\ker([\![\cdot]\!]) = \{\mathbf{0}\}$ and

$$\begin{aligned} \dim\langle E^{\complement}_{\overline{u}\,\overline{v}}\rangle^n &= |E^n| - |E_{\overline{u}\,\overline{v}}| = \left(n^2 - n\right) - (2n-2) = \\ &= (n-1)^2 - (n-1) = \left|E^{n-1}\right| = \dim\left\langle E^{n-1}\right\rangle^{n-1} . \end{aligned}$$

Lemma 2.10. *For any* $\boldsymbol{a} \in \mathbb{R}^{|E^n|}$ *it holds*

$$[\![\boldsymbol{a}]\!] \cdot \overline{\boldsymbol{x}}' = \boldsymbol{a} \cdot \overline{\boldsymbol{x}} - a_{\overline{u}\,\overline{v}} \overline{x}_{\overline{u}\,\overline{v}}$$

Proof. Expanding the scalar products

$$\begin{aligned}[\![\boldsymbol{a}]\!] \cdot \overline{\boldsymbol{x}}' &= \sum_{ab \in E^{n-1}} ([\![\boldsymbol{a}]\!])_{ab}\, \overline{x}'_{ab} = \sum_{ab \in E^{n-1}} a_{\rho(ab)} \overline{x}_{\rho(ab)} = \\ &= \sum_{uv \in E^{\mathtt{C}}_{\overline{u}\,\overline{v}}} a_{uv} \overline{x}_{uv} = \boldsymbol{a} \cdot \overline{\boldsymbol{x}} - \sum_{uv \in E_{\overline{u}\,\overline{v}}} a_{uv} \overline{x}_{uv} = \boldsymbol{a} \cdot \overline{\boldsymbol{x}} - a_{\overline{u}\,\overline{v}} \overline{x}_{\overline{u}\,\overline{v}}.\end{aligned}$$

□

Proposition 2.11. *Let* $\overline{\boldsymbol{x}}' \in \mathfrak{P}_2^{n-1}$ *and* $[\![\,\cdot\,]\!]$ *be defined as above. Then*

$$[\![\mathtt{ACT}^n_{\overline{\boldsymbol{x}}}]\!] = \mathtt{ACT}^{n-1}_{\overline{\boldsymbol{x}}'}$$

Proof. Let us begin from the space of boundary constraints. For all $ab \in E^{n-1}$ we can take $uv \in E^{\mathtt{C}}_{\overline{u}\,\overline{v}} = \rho\left(E^{n-1}\right)$ such that $uv = \rho(ab)$ and vice-versa, and directly from the definition of $[\![\,\cdot\,]\!]$ together with the fact that bnd^n_{uv} is the uv-th standard vector we obtain

$$[\![\mathrm{bnd}^n_{uv}]\!] = \sum_{ij \in E^{n-1}} \mathrm{bnd}^{n-1}_{ij} \cdot (\mathrm{bnd}^n_{uv})_{\rho(ij)} = \mathrm{bnd}^{n-1}_{ab} \cdot (\mathrm{bnd}^n_{uv})_{\rho(ab)} = \mathrm{bnd}^{n-1}_{ab}.$$

Also, we have $(\mathrm{bnd}^n_{uv})_{\overline{u}\,\overline{v}} = 0$ since $uv \in E^{\mathtt{C}}_{\overline{u}\,\overline{v}}$, therefore from Lemma 2.10 it holds

$$\mathrm{bnd}^n_{uv} \cdot \overline{\boldsymbol{x}} = \mathrm{bnd}^{n-1}_{ab} \cdot \overline{\boldsymbol{x}}'.$$

which makes it so that bnd^n_{uv} is active on $\overline{\boldsymbol{x}}$ if and only if bnd^{n-1}_{ab} is active on $\overline{\boldsymbol{x}}'$. If instead $uv \in E_{\overline{u}\,\overline{v}}$, it is easy to see that from equation (2.3) we would have

$$[\![\mathrm{bnd}^n_{uv}]\!] = \left[\!\left[\mathrm{pr}_{\langle E^{\mathtt{C}}_{\overline{u}\,\overline{v}} \rangle^n} (\mathrm{bnd}^n_{uv})\right]\!\right] = [\![\mathbf{0}]\!] = \mathbf{0}.$$

Summing it all up we therefore have

$$\left[\!\left[\mathtt{BND}^n_{E^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}\right]\!\right] = \left[\!\left[\mathtt{BND}^n_{E^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}}) \cap E^{\mathtt{C}}_{\overline{u}\,\overline{v}}}\right]\!\right] = \mathtt{BND}^{n-1}_{E^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')}.$$

With regards to out-degree constraints, we can write $\deg^{+,n}_u = \sum_{uv \in \delta^+_{\mathcal{K}_n}(u)} \mathrm{bnd}^n_{uv}$. Now, different possibilities arise:

- ▷ for $u = \overline{u}$, we have $\overline{u}v \in E_{\overline{u}\,\overline{v}}$ for all $\overline{u}v \in \delta^+_{\mathcal{K}_n}(\overline{u})$, thus
$$\left[\!\left[\deg^{+,n}_{\overline{u}}\right]\!\right] = \sum\nolimits_{\overline{u}v \in \delta^+_{\mathcal{K}_n}(\overline{u})} [\![\mathrm{bnd}^n_{\overline{u}v}]\!] = \mathbf{0};$$
- ▷ for $u = \overline{v}$, we have $\overline{v}\,\overline{u} \in E_{\overline{u}\,\overline{v}}$ and $\overline{v}v \in E^{\mathtt{C}}_{\overline{u}\,\overline{v}}$ for all $\overline{v}v \in \delta^+_{\mathcal{K}_n}(\overline{v}) \setminus \{\overline{v}\,\overline{u}\}$, thus
$$\left[\!\left[\deg^{+,n}_{\overline{v}}\right]\!\right] = \sum\nolimits_{\overline{v}v \in \delta^+_{\mathcal{K}_n}(\overline{v}) \setminus \{\overline{v}\,\overline{u}\}} [\![\mathrm{bnd}^n_{\overline{v}v}]\!] = \sum\nolimits_{\overline{w}v \in \delta^+_{\mathcal{K}_{n-1}}(\overline{w})} \mathrm{bnd}^{n-1}_{\overline{w}v} = \deg^{+,n-1}_{\overline{w}};$$
- ▷ for $u \in V^n \setminus \{\overline{u}, \overline{v}\}$, we have $u\overline{v} \in E_{\overline{u}\,\overline{v}}$ and $uv \in E^{\mathtt{C}}_{\overline{u}\,\overline{v}}$ for all $uv \in \delta^+_{\mathcal{K}_n}(u) \setminus \{u\overline{v}\}$, thus
$$[\![\deg^{+,n}_u]\!] = \sum\nolimits_{uv \in \delta^+_{\mathcal{K}_n}(u) \setminus \{u\overline{v}\}} [\![\mathrm{bnd}^n_{uv}]\!] = \sum\nolimits_{uv \in \delta^+_{\mathcal{K}_{n-1}}(\overline{w})} \mathrm{bnd}^{n-1}_{uv} = \deg^{+,n-1}_u.$$

In particular, $\forall\, a \in V^{n-1} \exists\, u \in V^n$ such that $\left[\!\left[\deg_u^{+,n}\right]\!\right] = \deg_a^{+,n-1}$. Therefore

$$\left[\!\left[\mathtt{DEG}^{+,n}_{V^n}\right]\!\right] = \mathtt{DEG}^{+,n-1}_{V^{n-1}}.$$

Similarly for the in-degree constraints, $\left[\!\left[\mathtt{DEG}^{-,n}_{V^n}\right]\!\right] = \mathtt{DEG}^{-,n-1}_{V^{n-1}}$.

Finally we study the subtour elimination constraints. Let $S \in \mathscr{S}^n, S^{\complement} \coloneqq V^n \setminus S$ and consider the equation $\mathrm{sep}^n_S = \sum_{u\in S}\sum_{v\in S^{\complement}} \mathrm{bnd}^n_{uv}$. Again, we have multiple possibilities.

▷ If $\{\overline{u},\overline{v}\} \subseteq S$, we have for $S' \coloneqq \{\overline{w}\} \cup S \setminus \{\overline{u},\overline{v}\}$ and $S'^{\complement} \coloneqq V^{n-1} \setminus S' = S^{\complement}$

$$\begin{aligned}
\left[\!\left[\mathrm{sep}^n_S\right]\!\right] &= \sum\nolimits_{v\in S^{\complement}} \left(\cancel{\left[\!\left[\mathrm{bnd}^n_{\overline{u}v}\right]\!\right]} + \left[\!\left[\mathrm{bnd}^n_{\overline{v}v}\right]\!\right] + \sum\nolimits_{u\in S\setminus\{\overline{u},\overline{v}\}} \left[\!\left[\mathrm{bnd}^n_{uv}\right]\!\right] \right) = \\
&= \sum\nolimits_{v\in S'^{\complement}} \left(\mathrm{bnd}^{n-1}_{\overline{w}v} + \sum\nolimits_{u\in S'\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{uv} \right) = \\
&= \sum\nolimits_{u\in S'} \sum\nolimits_{v\in S'^{\complement}} \mathrm{bnd}^{n-1}_{uv} = \begin{cases} \deg^{+,n-1}_{\overline{w}} & \text{if } S = \{\overline{u},\overline{v}\}\,, \\ \mathrm{sep}^{n-1}_{S'} & \text{otherwise.} \end{cases}
\end{aligned}$$

In the latter case $|S| > 2$ notice how since $\{\overline{u},\overline{v}\} \subset S$ there will be $(\mathrm{sep}^n_S)_{\overline{u}\,\overline{v}} = 0$, thus similar to the bnd^n_{uv} case Lemma 2.10 implies that $S \in \mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}}) \iff S' \in \mathscr{S}^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')$.

▷ If $\{\overline{u},\overline{v}\} \subseteq S^{\complement}$, we have for $S''^{\complement} = \{\overline{w}\} \cup S^{\complement} \setminus \{\overline{u},\overline{v}\}$ and $S'' \coloneqq V^{n-1} \setminus S''^{\complement} = S$

$$\begin{aligned}
\left[\!\left[\mathrm{sep}^n_S\right]\!\right] &= \sum\nolimits_{u\in S} \left(\left[\!\left[\mathrm{bnd}^n_{u\overline{u}}\right]\!\right] + \cancel{\left[\!\left[\mathrm{bnd}^n_{u\overline{v}}\right]\!\right]} + \sum\nolimits_{v\in S^{\complement}\setminus\{\overline{u},\overline{v}\}} \left[\!\left[\mathrm{bnd}^n_{uv}\right]\!\right] \right) = \\
&= \sum\nolimits_{u\in S''} \left(\mathrm{bnd}^{n-1}_{u\overline{w}} + \sum\nolimits_{v\in S''^{\complement}\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{uv} \right) = \\
&= \sum\nolimits_{u\in S''} \sum\nolimits_{v\in S''^{\complement}} \mathrm{bnd}^{n-1}_{uv} = \begin{cases} \deg^{-,n-1}_{\overline{w}} & \text{if } S^{\complement} = \{\overline{u},\overline{v}\}\,, \\ \mathrm{sep}^{n-1}_{S''} & \text{otherwise.} \end{cases}
\end{aligned}$$

Similarly to the previous case, $S \in \mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}}) \iff S'' \in \mathscr{S}^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')$.

▷ Let now $\overline{u} \in S$ and $\overline{v} \in S^{\complement}$, and assume also that $S \in \mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})$. We have for

$$S' \coloneqq S \setminus \{\overline{u}\} = V^{n-1} \setminus S'^{\complement}, \qquad S'^{\complement} \coloneqq \{\overline{w}\} \cup S^{\complement} \setminus \{\overline{v}\} = V^{n-1} \setminus S',$$

$$\begin{aligned}
\left[\!\left[\mathrm{sep}^n_S\right]\!\right] &= \sum_{u\in S\setminus\{\overline{u}\}} \sum_{v\in S^{\complement}\setminus\{\overline{v}\}} \left[\!\left[\mathrm{bnd}^n_{uv}\right]\!\right] + \sum_{v\in S^{\complement}\setminus\{\overline{v}\}} \cancel{\left[\!\left[\mathrm{bnd}^n_{\overline{u}v}\right]\!\right]} + \sum_{u\in S\setminus\{\overline{u}\}} \cancel{\left[\!\left[\mathrm{bnd}^n_{u\overline{v}}\right]\!\right]} + \cancel{\left[\!\left[\mathrm{bnd}^n_{\overline{u}\,\overline{v}}\right]\!\right]} = \\
&= \sum_{u\in S'} \sum_{v\in S'^{\complement}\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{uv} + \left[\sum\nolimits_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} - \sum\nolimits_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} \right] = \\
&= \sum_{u\in S'} \sum_{v\in S'^{\complement}} \mathrm{bnd}^{n-1}_{uv} - \sum\nolimits_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} = \\
&= \begin{cases} \deg^{+,n-1}_{u'} - \sum_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} & \text{if } S = \{\overline{u},u'\}\,, \\ \mathrm{sep}^{n-1}_{S'} - \sum_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} & \text{otherwise.} \end{cases}
\end{aligned}$$

Let us assume that the latter is the case (the former analogously leads to the same result since $\deg^{+,n-1}_{u'} \in \mathtt{DEG}^{+,n-1}_{V^{n-1}}$). We want to prove that $\sum_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} \cdot \overline{\boldsymbol{x}}' = \sum_{u\in S'} \overline{x}'_{u\overline{w}} = 1$. Given that we assumed $S \in \mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})$, we have $\mathrm{sep}^n_S \cdot \overline{\boldsymbol{x}} = 1$ and

$$1 = \mathrm{sep}^n_S \cdot \overline{\boldsymbol{x}} = \sum\nolimits_{u\in S\setminus\{\overline{u}\}} \sum\nolimits_{v\in S^{\complement}} \overline{x}_{uv} + \sum\nolimits_{v\in S^{\complement}} \overline{x}_{\overline{u}v} = \sum\nolimits_{u\in S\setminus\{\overline{u}\}} \sum\nolimits_{v\in S^{\complement}} \overline{x}_{uv} + 1,$$

since $\sum_{v\in S^{\complement}} \overline{x}_{\overline{u}v} = \overline{x}_{\overline{u}\,\overline{v}} = 1$, therefore $\sum_{u\in S\setminus\{\overline{u}\}} \sum_{v\in S^{\complement}} \overline{x}_{uv} = 0$. Consider now

$$\mathrm{sep}^n_{S\setminus\{\overline{u}\}} \cdot \overline{\boldsymbol{x}} = \cancel{\sum\nolimits_{u\in S\setminus\{\overline{u}\}} \sum\nolimits_{v\in S^{\complement}} \overline{x}_{uv}} + \sum\nolimits_{u\in S\setminus\{\overline{u}\}} \overline{x}_{u\overline{u}} = \sum\nolimits_{u\in S\setminus\{\overline{u}\}} \overline{x}_{u\overline{u}}.$$

While $\sum_{u\in S\setminus\{\overline{u}\}} \overline{x}_{u\overline{u}} \le \sum_{u\in V^n\setminus\{\overline{u}\}} \overline{x}_{u\overline{u}} = \deg^{-,n}_{\overline{u}} = 1$ by the subtour elimination constraints it holds $\mathrm{sep}^n_{S\setminus\{\overline{u}\}} \cdot \overline{\boldsymbol{x}} \ge 1$ therefore from Lemma 2.10

$$1 = \sum\nolimits_{u\in S\setminus\{\overline{u}\}} \overline{x}_{u\overline{u}} = \sum\nolimits_{u\in S\setminus\{\overline{u}\}} [\![\mathrm{bnd}^n_{u\overline{u}}]\!] \cdot \overline{\boldsymbol{x}} = \sum\nolimits_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} \cdot \overline{\boldsymbol{x}}'.$$

This in turn implies that $S' \in \mathscr{S}^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')$ since again from Lemma 2.10 we have

$$\mathrm{sep}^{n-1}_{S'} \cdot \overline{\boldsymbol{x}}' = [\![\mathrm{sep}^n_S]\!] \cdot \overline{\boldsymbol{x}}' + \sum\nolimits_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} \cdot \overline{\boldsymbol{x}}' = \mathrm{sep}^n_S \cdot \overline{\boldsymbol{x}} - (\mathrm{sep}^n_S)_{\overline{u}\,\overline{v}} \cdot \overline{x}_{\overline{u}\,\overline{v}} + 1 = 1,$$

because both $(\mathrm{sep}^n_S)_{\overline{u}\,\overline{v}} = 1$ and $\overline{x}_{\overline{u}\,\overline{v}} = 1$. It also implies that $\mathrm{bnd}^{n-1}_{u\overline{w}}$ is active $\forall\, u \in S'^{\complement} \setminus \{\overline{w}\}$, since

$$\sum_{u\in S'^{\complement}\setminus\{\overline{w}\}} 4\overline{x}'_{u\overline{w}} = \sum_{u\in\delta^-_{\mathcal{K}_{n-1}}(\overline{w})} 4\overline{x}'_{u\overline{w}} - \sum_{u\in S'} 4\overline{x}_{u\overline{w}} = 1-1 = 0 \quad\Longrightarrow\quad \overline{x}'_{u\overline{w}} = 0 \quad \forall\, u \in S'^{\complement}\setminus\{\overline{w}\}\,.$$

from which $\sum_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}} \in \mathtt{DEG}^{-,n-1}_{V^{n-1}} + \mathtt{BND}^{n-1}_{E^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')}$ and therefore $[\![\mathrm{sep}^n_S]\!] \in \mathtt{ACT}^{n-1}_{\overline{\boldsymbol{x}}'}$.

▷ Analogously for $\overline{v} \in S$ and $\overline{u} \in S^{\complement}$ again assuming $S \in \mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})$ we have for

$$S'' \coloneqq \{\overline{w}\} \cup S \setminus \{\overline{v}\} = V^{n-1} \setminus S''^{\complement}, \qquad S''^{\complement} \coloneqq S^{\complement} \setminus \{\overline{u}\} = V^{n-1} \setminus S'',$$

$$\begin{aligned}
[\![\mathrm{sep}^n_S]\!] &= \sum_{u\in S\setminus\{\overline{v}\}} \sum_{v\in S^{\complement}\setminus\{\overline{u}\}} [\![\mathrm{bnd}^n_{uv}]\!] + \sum_{v\in S^{\complement}\setminus\{\overline{u}\}} [\![\mathrm{bnd}^n_{\overline{v}v}]\!] + \sum_{u\in S\setminus\{\overline{v}\}} [\![\mathrm{bnd}^n_{u\overline{u}}]\!] + \cancel{[\![\mathrm{bnd}^n_{\overline{v}\,\overline{u}}]\!]} = \\
&= \sum_{u\in S''\setminus\{\overline{w}\}} \sum_{v\in S''^{\complement}} \mathrm{bnd}^{n-1}_{uv} + \sum_{v\in S''^{\complement}} \mathrm{bnd}^{n-1}_{\overline{w}v} + \sum_{u\in S''\setminus\{\overline{w}\}} \mathrm{bnd}^n_{u\overline{w}} = \\
&= \sum_{u\in S'} \sum_{v\in S''^{\complement}} \mathrm{bnd}^{n-1}_{uv} + \sum\nolimits_{u\in S''\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{u\overline{w}} = \\
&= \begin{cases} \deg^{-,n-1}_{v'} + \sum_{u\in S''\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{u\overline{w}} & \text{if } S^{\complement} = \{\overline{u}, v'\}\,, \\ \mathrm{sep}^{n-1}_{S''} + \sum_{u\in S''\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{u\overline{w}} & \text{otherwise.} \end{cases}
\end{aligned}$$

Let us once more assume without loss of generality that the latter is the case. Notice how this result is complementary to the last one in the sense that

$$\begin{aligned}
[\![\mathrm{sep}^n_S]\!] - [\![\mathrm{sep}^n_{S^{\complement}}]\!] &= \Big(\mathrm{sep}^{n-1}_{S''} + \sum\nolimits_{u\in S''\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{u\overline{w}}\Big) - \Big(\mathrm{sep}^{n-1}_{S'} - \sum\nolimits_{u\in S'} \mathrm{bnd}^{n-1}_{u\overline{w}}\Big) = \\
&= \mathrm{sep}^{n-1}_{S''} - \mathrm{sep}^{n-1}_{S'} + \deg^{-,n-1}_{\overline{w}}
\end{aligned}$$

with $S' = S''^{\complement}$. Thus from the previous point we have $\sum_{u\in S''^{\complement}} \mathrm{bnd}^{n-1}_{u\overline{w}} \cdot \overline{\boldsymbol{x}}' = 1$ again obtaining $\sum_{u\in S''\setminus\{\overline{w}\}} \overline{x}'_{u\overline{w}} = 0$ which implies $\sum_{u\in S''\setminus\{\overline{w}\}} \mathrm{bnd}^{n-1}_{u\overline{w}} \in \mathtt{BND}^{n-1}_{E^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')}$, and once more from Lemma 2.10 $\mathrm{sep}^{n-1}_{S'}$ is active, therefore giving $[\![\mathrm{sep}^n_S]\!] \in \mathtt{ACT}^{n-1}_{\overline{\boldsymbol{x}}'}$.

Having finished our exhaustive (and exhausting) breakdown of the possibilities, we are able to affirm that

$$\left[\!\left[\mathtt{SEP}^{n}_{\mathscr{S}^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}\right]\!\right] \subseteq \mathtt{ACT}^{n-1}_{\overline{\boldsymbol{x}}'}.$$

But since $\forall\, S' \in \mathscr{S}^{n-1}$ we are able to construct a set $S \subseteq V^n$ such that

$$[\![\mathrm{sep}^{n}_{S}]\!] = \mathrm{sep}^{n-1}_{S'}, \qquad \text{with } S \coloneqq \begin{cases} \{\overline{u},\overline{v}\} \cup S' \setminus \{\overline{w}\} & \text{if } \overline{w} \in S', \\ S' & \text{if } \overline{w} \notin S', \end{cases}$$

this gives us the guarantee that $\mathtt{SEP}^{n-1}_{\mathscr{S}^{n-1}} \subseteq \left[\!\left[\mathtt{SEP}^{n}_{\mathscr{S}^{n}}\right]\!\right]$ and $\mathtt{SEP}^{n-1}_{\mathscr{S}^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')} \subseteq \left[\!\left[\mathtt{SEP}^{n}_{\mathscr{S}^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}\right]\!\right]$. Thus all in all we have

$$\begin{aligned}[\![\mathtt{ACT}^{n}_{\overline{\boldsymbol{x}}}]\!] &= \left[\!\left[\mathtt{DEG}^{+,n}_{V^n}\right]\!\right] + \left[\!\left[\mathtt{DEG}^{-,n}_{V^n}\right]\!\right] + \left[\!\left[\mathtt{SEP}^{n}_{\mathscr{S}^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}\right]\!\right] + \left[\!\left[\mathtt{BND}^{n}_{E^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}\right]\!\right] = \\ &= \mathtt{DEG}^{+,n-1}_{V^{n-1}} + \mathtt{DEG}^{-,n-1}_{V^{n-1}} + \mathtt{SEP}^{n-1}_{\mathscr{S}^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')} + \mathtt{BND}^{n-1}_{E^{n-1}_{\mathtt{ACT}}(\overline{\boldsymbol{x}}')} = \mathtt{ACT}^{n-1}_{\overline{\boldsymbol{x}}'}.\end{aligned}$$

□

Direct consequence of this result is that it allows us to solve the problem of the extremality check on pure half-integer solutions only, $\overline{\boldsymbol{x}} \in \mathfrak{P}^{n-1}_{2} \cap \{0, 1/2\}^{|E^n|}$, as any general solution can by brought in this form by iteratively "collapsing" its 1-arcs.

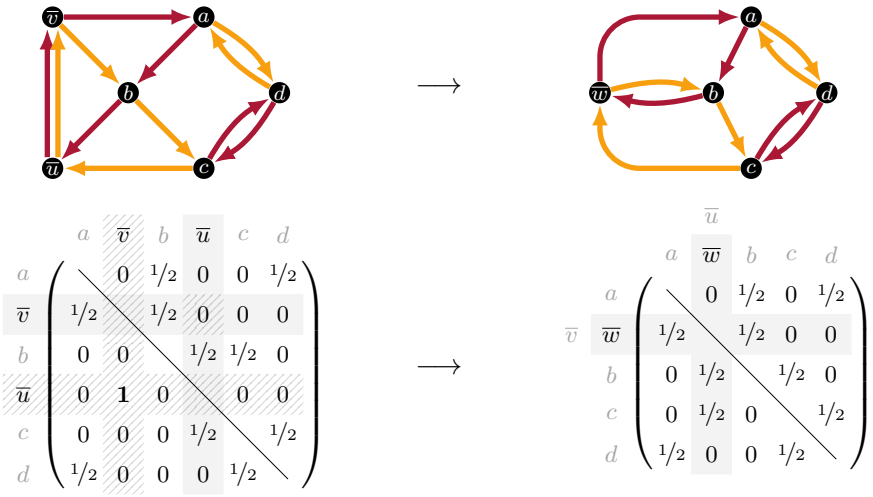


Figure 2.5: Example of the effect of identifying a 1-arc to a single point, equivalent to focusing on the subspace $\langle E^{\mathfrak{C}}_{\overline{u}\,\overline{v}}\rangle^n$.

Subspace $\langle \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})\rangle^n$

Finally, we see that it is enough to verify that the vectors $\deg^{+,n}_{u}, \deg^{-,n}_{v}, \mathrm{sep}^{n}_{S}$ for all $u, v \in V^n, S \in \mathscr{S}^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})$ generate the subspace identified by the arcs in $\mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})$ to know that the active constraints are full-rank. This is formally expressed by the following.

Proposition 2.12. *The condition* $\mathtt{ACT}^{n}_{\overline{\boldsymbol{x}}} = \mathbb{R}^{|E^n|}$ *of (2.2) is equivalent to*

$$\mathrm{pr}_{\langle \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})\rangle^n}\left(\mathtt{DEG}^{+,n}_{V^n} + \mathtt{DEG}^{-,n}_{V^n} + \mathtt{SEP}^{n}_{\mathscr{S}^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}\right) = \langle \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})\rangle^n.$$

Proof. By construction $\mathbb{R}^{|E^n|} = \mathtt{BND}^{n}_{E^n} = \mathtt{BND}^{n}_{E^{n}_{\mathtt{ACT}}(\overline{\boldsymbol{x}})} \oplus \mathtt{BND}^{n}_{\mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})}$.

□

Circuit Representation

We are now ready to proceed with designing our algorithm. Let us notate $E^n_{\overline{\boldsymbol{x}}} \coloneqq \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})$. Thanks to what has been proved up to here, we are safe to assume that $\overline{\boldsymbol{x}}$ is made out of only $0, 1/2$ entries ($\overline{\boldsymbol{x}} \in \mathfrak{P}_2^{n-1} \cap \{0, 1/2\}^{|E^n|}$) and to consider only the rank maximality over the subspace $\langle E^n_{\overline{\boldsymbol{x}}} \rangle^n$, thus in the reminder we will safely assume that all vectors $\boldsymbol{a} \in \mathtt{DEG}^n_{V^n} \cup \mathtt{SEP}^n_{\mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}$ are replaced with their projection $\lfloor \boldsymbol{a} \rfloor \coloneqq \mathrm{pr}_{\langle E^n_{\overline{\boldsymbol{x}}} \rangle^n}(\boldsymbol{a})$ (by construction the bound vectors $\mathtt{BND}^n_{E^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})}$ are all projected to $\mathbf{0}$ and can be discarded).

Let the following sets of projected vectors be defined,

$$\mathrm{Deg}^+_{\overline{\boldsymbol{x}}} \coloneqq \{\lfloor \mathrm{deg}^{+,n}_u \rfloor \mid u \in V^n\}, \qquad \mathrm{Deg}_{\overline{\boldsymbol{x}}} \coloneqq \mathrm{Deg}^+_{\overline{\boldsymbol{x}}} \cup \mathrm{Deg}^-_{\overline{\boldsymbol{x}}},$$
$$\mathrm{Deg}^-_{\overline{\boldsymbol{x}}} \coloneqq \{\lfloor \mathrm{deg}^{-,n}_v \rfloor \mid v \in V^n\}, \qquad \mathrm{Sep}_{\overline{\boldsymbol{x}}} \coloneqq \{\lfloor \mathrm{sep}^n_S \rfloor \mid S \in \mathscr{S}^n_{\mathtt{ACT}}(\overline{\boldsymbol{x}})\}.$$

Since by hypothesis $\forall\, \boldsymbol{a} \in \mathrm{Deg}_{\overline{\boldsymbol{x}}} \cup \mathrm{Sep}_{\overline{\boldsymbol{x}}}$ we have $\boldsymbol{a} \cdot \overline{\boldsymbol{x}} = 1$, keeping in mind that $\overline{x}_{uv} \in \{0, 1/2\}$ and $a_{uv} \in \{0, 1\}$ for all $uv \in E^n$, it follows

$$|\mathrm{supp}_{E^n} \lfloor \boldsymbol{a} \rfloor| = |\mathrm{supp}_{E^n}(\boldsymbol{a}) \cap \mathrm{supp}_{E^n}(\overline{\boldsymbol{x}})| = |\mathrm{supp}_{E^n}(\boldsymbol{a} \cdot \overline{\boldsymbol{x}})| = \boldsymbol{a} \cdot 2\overline{\boldsymbol{x}} = 2,$$

showing how $\lfloor \boldsymbol{a} \rfloor$ is always a $0,1$-vector in $\langle E^n_{\overline{\boldsymbol{x}}} \rangle^n$ with exactly two 1-components. If $\mathrm{supp}_{E^n} \lfloor \boldsymbol{a} \rfloor = \{u_1v_1, u_2v_2\}$ we will write $[u_1v_1|u_2v_2] = [u_2v_2|u_1v_1] \coloneqq \lfloor \boldsymbol{a} \rfloor$ and will refer to u_1v_1 and u_2v_2 as the two *extremities* of $\lfloor \boldsymbol{a} \rfloor$ (notice also how all out- and in-degree vectors will be respectively in the form $[uv_1|uv_2]$ and $[u_1v|u_2v]$). Finally, unraveling the definitions we can write

$$[u_1v_1|u_2v_2] = \lfloor \mathrm{bnd}^n_{u_1v_1} \rfloor + \lfloor \mathrm{bnd}^n_{u_2v_2} \rfloor = \mathrm{bnd}^n_{u_1v_1} + \mathrm{bnd}^n_{u_2v_2}.$$

Let us focus first on the structure of $\mathrm{Deg}_{\overline{\boldsymbol{x}}}$: we see that for any $uv \in E^n_{\overline{\boldsymbol{x}}}$ there are exactly two degree vectors with non-null $\overline{u}\,\overline{v}$-component, that is $\lfloor \mathrm{deg}^{+,n}_{\overline{u}} \rfloor = [\overline{u}\,\overline{v}|\overline{u}v]$ and $\lfloor \mathrm{deg}^{-,n}_{\overline{v}} \rfloor = [\overline{u}\,\overline{v}|u\overline{v}]$; such fact justifies the following definitions.

Definition 2.13 (link, circuit, circuit partition, dual)**.** Let $\sim_A$ be the transitive closure of the homogeneous binary relation $\leftrightarrow_A$ on $E^n_{\overline{\boldsymbol{x}}}$ defined by $u_1v_1 \leftrightarrow_A u_2v_2$ if and only if

$$\exists\, \overline{u}\,\overline{v} \in E^n_{\overline{\boldsymbol{x}}} \quad \text{such that } [u_1v_1|\overline{u}\,\overline{v}], [\overline{u}\,\overline{v}|u_2v_2] \in A, \qquad \mathrm{Deg}_{\overline{\boldsymbol{x}}} \subseteq A \subseteq \mathrm{Deg}_{\overline{\boldsymbol{x}}} \cup \mathrm{Sep}_{\overline{\boldsymbol{x}}}.$$

Additionally, $\leftrightarrow_A$ and $\sim_A$ are trivially reflexive and symmetrical, thus $\sim_A$ is an equivalence relation. For any $u_1v_1 \neq u_2v_2$ we refer to $\overline{u}\,\overline{v}$ as the *link* between u_1v_1 and u_2v_2 and we write $u_1v_1 \overset{\overline{u}\,\overline{v}}{\longleftrightarrow}_A u_2v_2$, and we refer to the set of arcs that are linked to any other by $\overline{u}\,\overline{v}$ as $\mathrm{adj}_A(\overline{u}\,\overline{v}) \coloneqq \{uv \in A \mid \exists\, u'v' \in A, uv \overset{\overline{u}\,\overline{v}}{\longleftrightarrow}_A u'v'\}$ [2].

Let the *circuit partition* $\mathscr{L}_A$ be the partitioning of the arcs in $E^n_{\overline{\boldsymbol{x}}}$ given by the equivalence classes of $\sim_A$, and call *circuits* its elements $L \in \mathscr{L}_A$. Since $\forall\, u_1v_1, u_2v_2, u_3v_3 \in L$

$$u_1v_1 \overset{\overline{u}_1\overline{v}_1}{\longleftrightarrow}_A u_2v_2 \overset{\overline{u}_2\overline{v}_2}{\longleftrightarrow}_A u_3v_3 \quad \Longrightarrow \quad \overline{u}_1\overline{v}_1 \overset{u_2v_2}{\longleftrightarrow}_A \overline{u}_2\overline{v}_2,$$

it follows that there exists a $L' \in \mathscr{L}_A$ containing both $\overline{u}_1\overline{v}_1$ and $\overline{u}_2\overline{v}_2$; we refer to such L' as the *dual* of L denoted as $\overline{L}^A$, and refer to the set $\{L, \overline{L}^A\}$ as a *circuit pair*. From the definition it follows that $\overline{\overline{L}^A}^A = L$. We say that a $L \in \mathscr{L}_A$ for which $\overline{L}^A = L$ is *short-circuited* or simply *shorted*.

To conclude, notice also how $A \subseteq A'$ implies $\leftrightarrow_A \subseteq \leftrightarrow_{A'}$ and $\sim_A \subseteq \sim_{A'}$ and thus makes $\mathscr{L}_{A'}$ a coarser partition than $\mathscr{L}_A$. Finally, all the subscripts A of $\leftrightarrow_A$, $\sim_A$, adj_A, $\mathscr{L}_A$ $\overline{\cdot}^A$ will be dropped when it is unambiguous to do so.

[2] When $A = \mathrm{Deg}_{\overline{\boldsymbol{x}}}$, this notation is coherent with the arc-adjacency notion on the digraph $\mathscr{X}^n_{\overline{\boldsymbol{x}}}$.

Remark 2.14. Let us now focus on the case $A = \mathrm{Deg}_{\overline{\boldsymbol{x}}}$, the most interesting to us. Again, since any $\overline{u}\,\overline{v} \in E^n_{\overline{\boldsymbol{x}}}$ is the extremity of an out-degree $[\overline{u}\,\overline{v}|\overline{u}v]$ and an in-degree $[\overline{u}\,\overline{v}|u\overline{v}]$ vector we have that every $\overline{u}\,\overline{v} \in E^n_{\overline{\boldsymbol{x}}}$:

▷ links two distinct arcs $\overline{u}v, u\overline{v}$ via an out-degree and an in-degree vector;

▷ is linked to two (eventually equal) arcs $u'v, uv'$ by $\overline{u}v, u\overline{v}$ respectively.

As such it is implied that any circuit pair $\{L, \overline{L}\}$ identifies a succession of linked arcs and their links that eventually loops on itself (due to the finiteness of the set of arcs)

$$\rightarrow u_1v_1 \overset{\overline{u}_1\overline{v}_1}{\longleftrightarrow} u_2v_2 \overset{\overline{u}_2\overline{v}_2}{\longleftrightarrow} u_3v_3 \longleftarrow \cdots \longrightarrow u_{|L|}v_{|L|} \overset{\overline{u}_{|L|}\overline{v}_{|L|}}{\longleftarrow}$$

with $u_iv_i \in L, \overline{u}_i\overline{v}_i \in \overline{L}$ for all $i = \{1, ..., |L|\}$. This property firstly shows how $|L| = \left|\overline{L}\right|$. By taking into account the resulting "domino" succession of vectors

$$\Big([u_1v_1|\overline{u}_1\overline{v}_1], [\overline{u}_1\overline{v}_1|u_2v_2], [u_2v_2|\overline{u}_2\overline{v}_2], \ldots, [u_{|L|}v_{|L|}|\overline{u}_{|L|}\overline{v}_{|L|}], [\overline{u}_{|L|}\overline{v}_{|L|}|u_1v_1] \Big), \quad (2.4)$$

it follows that it must be composed of alternating out- and in-degree constraint vectors. But this in turn implies the non-trivial definition of the dual of a circuit: we have that L is not short-circuited ($L \neq \overline{L}$), since if it where $u_iv_i = \overline{u}_j\overline{v}_j$ for some i, j we would have either $[u_iv_i|\overline{u}_i\overline{v}_i], [u_jv_j|\overline{u}_j\overline{v}_j]$ or $[\overline{u}_j\overline{v}_j|u_{j+1}v_{j+1}], [\overline{u}_{i-1}\overline{v}_{i-1}|u_iv_i]$ (where the subscript is adjusted modulo $|L|$) as a pair of both out- or in-degree vectors with the same arc as extremity, against the hypothesis. Finally, the representation of L and $\overline{L}$ as a succession provides a fair justification for the names "circuit" and "link".

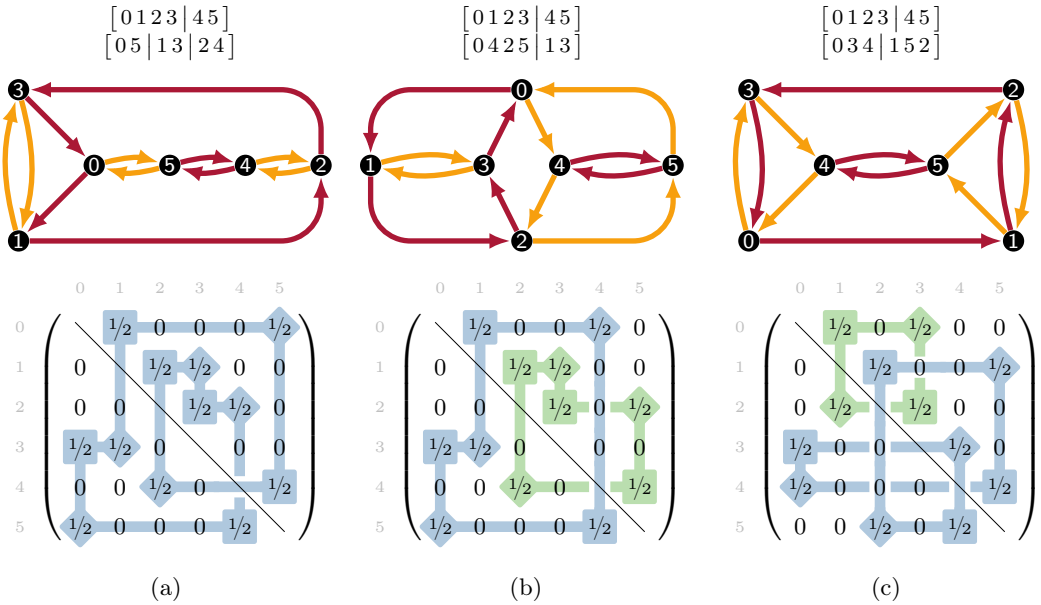


Figure 2.6: Examples of circuit partitions for $A = \mathrm{Deg}_{\overline{\boldsymbol{x}}}$ on the matrix representations of the solution's characteristic vectors; circuit pairs are displayed as closed curves of different colors, with duals represented as square/diamond nodes.

For $L, \overline{L} \in \mathscr{L}_A$ let us notate as $[L]_A$ (again dropping the subscript when unambiguous) the subset of constraints of A that constitute the linking of L and $\overline{L}$, formally

$$[L]_A \coloneqq \left\{ [uv|\overline{u}\,\overline{v}] \in A \;\middle|\; uv \in L, \overline{u}\,\overline{v} \in \overline{L} \right\}.$$

With regards to the subspaces $\operatorname{span}[L]_A$ generated by the circuits we notice that distinct circuit pairs generate disjoint subspaces, as the vectors $[L]_A$ have extremities (and thus support) entirely contained in $L \cup \overline{L}$ and the circuits themselves are a partition of $E^n_{\overline{\boldsymbol{x}}}$.

Returning once more to the case of $A = \mathrm{Deg}_{\overline{\boldsymbol{x}}}$ we can see that $[L]_{\mathrm{Deg}_{\overline{\boldsymbol{x}}}}$ is equivalent to the set expressed in equation (2.4), and we can thus decompose it as

$$[L]_{\mathrm{Deg}_{\overline{\boldsymbol{x}}}} = [L]^+ \cup [L]^-, \qquad \text{with } \begin{cases} [L]^+ \coloneqq [L]_{\mathrm{Deg}_{\overline{\boldsymbol{x}}}} \cap \mathrm{Deg}^+_{\overline{\boldsymbol{x}}}, \\ [L]^- \coloneqq [L]_{\mathrm{Deg}_{\overline{\boldsymbol{x}}}} \cap \mathrm{Deg}^-_{\overline{\boldsymbol{x}}}. \end{cases}$$

Under this assumptions we can observe that $\operatorname{span}[L]_{\mathrm{Deg}_{\overline{\boldsymbol{x}}}}$ has dimension $2|L| - 1$, since $\dim\left(\operatorname{span}[L]^+\right) = \dim\left(\operatorname{span}[L]^-\right) = |L|$ (both $[L]^+$ and $[L]^-$ are sets of linearly independent vectors) and $\dim\left(\operatorname{span}\left([L]^+ \cap [L]^-\right)\right) = 1$ since

$$\begin{aligned} \operatorname{span}\left([L]^+ \cap [L]^-\right) &= \operatorname{span}\left\{\textstyle\sum_{[e_1|e_2]\in[L]^+} [e_1|e_2]\right\} = \operatorname{span}\left\{\textstyle\sum_{[e'_2|e'_1]\in[L]^-} [e'_2|e'_1]\right\} = \\ &= \operatorname{span}\left\{\textstyle\sum_{e\in L\cup\overline{L}} \mathrm{bnd}^n_e\right\}. \end{aligned}$$

We would therefore aim at finding a subtour elimination constraint in $\mathrm{Sep}_{\overline{\boldsymbol{x}}}$ that "fills" the remaining degree of freedom of $\operatorname{span}[L]_{\mathrm{Deg}_{\overline{\boldsymbol{x}}}}$. Fortunately, a characterization of those "filling" constraints is given by the following propositions.

Proposition 2.15 (circuits merging)**.** *Let $[e_1|e_2] \in \mathrm{Sep}_{\overline{\boldsymbol{x}}}$ and define $A' \coloneqq A \cup \{[e_1|e_2]\}$ for a given A as in definition 2.13. Given $L_1, L_2 \in \mathscr{L}_A$ such that $e_1 \in L_1, e_2 \in L_2$, then*

$$\mathscr{L}_{A'} = \mathscr{L}' \cup \mathscr{L}_A \setminus \left\{L_1, L_2, \overline{L_1}^A, \overline{L_2}^A\right\},$$

with:

1. *$\mathscr{L}' \coloneqq \left\{L_1 \cup \overline{L_1}^A \cup L_2 \cup \overline{L_2}^A\right\}$ if $L_1 = L_2$, $L_1 = \overline{L_1}^A$ or $L_2 = \overline{L_2}^A$, where the only circuit in $\mathscr{L}'$ is shorted;*
2. *$\mathscr{L}' = \left\{L_1 \cup \overline{L_2}^A, \overline{L_1}^A \cup L_2\right\}$ otherwise, where the two circuits in $\mathscr{L}'$ are duals and not shorted;*

Proof. We already remarked how $\leftrightarrow_A \subseteq \leftrightarrow_{A'}$, and we now construct explicitly the elements in $\leftrightarrow_{A'} \setminus \leftrightarrow_A$. We observe that the element $[e_1|e_2]$ can only be involved in the linkage of one of its extremities to arcs adjacent (with respect to A) to the other one, that is

$$e_1 \overset{[e_1|e_2]}{\longleftrightarrow}_{A'} \overline{e}_2 \quad \forall\, \overline{e}_2 \in \operatorname{adj}_A(e_2) \qquad \text{and} \qquad e_2 \overset{[e_1|e_2]}{\longleftrightarrow}_{A'} \overline{e}_1 \quad \forall\, \overline{e}_1 \in \operatorname{adj}_A(e_1).$$

Since $\operatorname{adj}_A(e_2) \neq \emptyset$ (remark 2.14) and $\operatorname{adj}_A(e_2) \subseteq \overline{L_2}^A$, the two equivalence classes L_1 and $\overline{L_2}^A$ having some A'-linked representatives thus merge under the new equivalence relation $\sim_{A'}$. Similarly happens for L_2 and $\overline{L_1}^A$. This means that the four (eventually non-distinct) $\sim_A$-equivalence classes merge into the $\sim_{A'}$-equivalence class(es):

1. $\mathscr{L}' = \left\{L_1 \cup \overline{L_1}^A \cup L_2 \cup \overline{L_2}^A\right\}$ when either $L_1 = L_2$, $L_1 = \overline{L_1}^A$ or $L_2 = \overline{L_2}^A$ ($\overline{L_1}^A = \overline{L_2}^A$ is equivalent to $L_1 = L_2$); in this case the only circuit $L \in \mathscr{L}'$ it its own dual $\overline{L}^{A'} = L$ and L is thus shorted;
2. $\mathscr{L}' = \left\{L_1 \cup \overline{L_2}^A,\ \overline{L_1}^A \cup L_2\right\}$ otherwise; the dualism between the two circuits $\{L, L'\} = \mathscr{L}'$ comes from $e_1 \in L$ and $e_2 \in L'$ becoming duals by construction, while $L \neq L' = \overline{L}^{A'}$ since it would imply the satisfaction of the case (1.) condition.

Since this exhaust the set $\leftrightarrow_{A'} \setminus \leftrightarrow_A$, no other class of $\mathscr{L}_A$ gets merged under $\sim_{A'}$. □

Corollary 2.16. *In the above notation, if $L_2 = \overline{L_1}^A$, then $\mathscr{L}_{A'} = \mathscr{L}_A$.*

Proof. $\mathscr{L}' = \left\{L_1, \overline{L_1}^A\right\}$, therefore $\mathscr{L}_{A'} = \left\{L_1, \overline{L_1}^A\right\} \cup \mathscr{L}_A \setminus \left\{L_1, \overline{L_1}^A\right\} = \mathscr{L}_A$. □

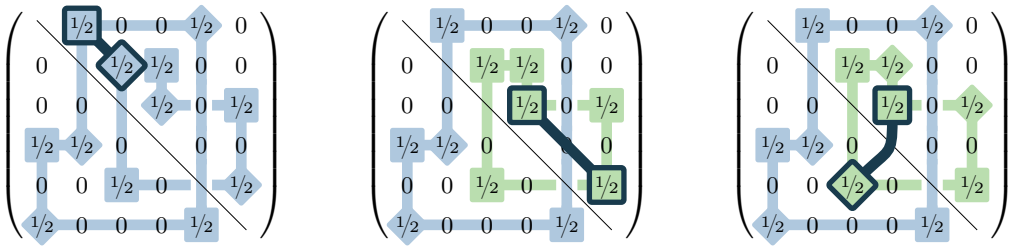


Figure 2.7: Schematic representation of different possible circuits merging in the example of Figure 2.6b; in the notation of Proposition 2.15, the dark-colored connections represents the newly added vector $[e_1|e_2]$ with $e_1 \in L_1, e_2 \in L_2$, while the partitions resulting from the merging are represented similarly to Figure 2.6, with shorted partitions shown having only square or only diamond nodes.

Lemma 2.17. *Let $L \in \mathscr{L}_A$ such that $\dim\left(\operatorname{span}[L]_A\right) = 2|L| - 1$ and $[e_1|e_2] \in \operatorname{Sep}_{\overline{\boldsymbol{x}}} \setminus A$ such that $e_1 \in L$. Then $[e_1|e_2]$ is linearly independent from $[L]_A$ if and only if $e_2 \notin \overline{L}^A$.*

Proof. Let us consider the linear application

$$\sigma_L^A \colon \langle E^n_{\overline{\boldsymbol{x}}} \rangle^n \to \mathbb{R}, \qquad \sigma_L^A(\boldsymbol{a}) = \textstyle\sum_{e \in L} a_e - \sum_{e \in \overline{L}^A} a_e.$$

It is easy to see from the definition of $[L]_A$ that $\sigma_L^A([e_1'|e_2']) = 0$ for all $[e_1'|e_2'] \in [L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}$, thus $[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}} \subseteq \ker \sigma_L^A$; but since we have that $\dim\left(\ker \sigma_L^A\right) = 2|L| - 1$ and we already know that $\dim\left(\operatorname{span}[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}\right) = 2|L| - 1$, it must be $\operatorname{span}[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}} = \ker \sigma_L^A$. With regards to the vector $[e_1|e_2]$, remembering that $e_1 \in L$, we have

$$\sigma_L^A([e_1|e_2]) = \begin{cases} 2 & \text{if } e_2 \in L, \\ 1 & \text{if } e_2 \notin L \cup \overline{L}^A, \\ 0 & \text{if } e_2 \in \overline{L}^A. \end{cases}$$

In the last case $[e_1|e_2] \in \ker \sigma_L^A = \operatorname{span}[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}$, while in the first two $[e_1|e_2] \notin \operatorname{span}[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}$.

For any $\operatorname{Deg}_{\overline{\boldsymbol{x}}} \subseteq A \subsetneq \operatorname{Deg}_{\overline{\boldsymbol{x}}} \cup \operatorname{Sep}_{\overline{\boldsymbol{x}}}$ it holds $[L]_A = [L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}} \cup [L]_{A \setminus \operatorname{Deg}_{\overline{\boldsymbol{x}}}}$, but since

$$\dim\left(\operatorname{span}[L]_A\right) = \dim\left(\operatorname{span}[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}\right) = 2|L| - 1,$$

the linear dependency of $[e_1|e_2]$ from $[L]_A$ boils down to its dependency from $[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}$. □

Proposition 2.18. *Let $L \in \mathscr{L}_A$. Then* $\operatorname{span}[L]_A$ *is equal to* $|L|$ *if* L *is shorted and to* $2|L|-1$ *otherwise.*

Proof. By writing $A = \operatorname{Deg}_{\overline{\boldsymbol{x}}} \cup \{\boldsymbol{a}^1, \dots, \boldsymbol{a}^m\}$ with $\boldsymbol{a}^k \in \operatorname{Sep}_{\overline{\boldsymbol{x}}}$ it can be seen that $\mathscr{L}_A$ is obtained from $\mathscr{L}_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}$ by iteratively merging the vectors $(\boldsymbol{a}^1, \dots, \boldsymbol{a}^m)$ as described in proposition Proposition 2.15. We thus prove this proposition by induction on the sequence of sets $(A_0, A_1, \dots, A_m)$ with $A_k \coloneqq \operatorname{Deg}_{\overline{\boldsymbol{x}}} \cup \bigcup_{i=1}^{k} \boldsymbol{a}^k$, $k = \{1, ..., m\}$ and $A_0 \coloneqq \operatorname{Deg}_{\overline{\boldsymbol{x}}}$. The base case $A_0 = \operatorname{Deg}_{\overline{\boldsymbol{x}}}$, where $L \in \mathscr{L}_{A_0}$ is always non-shorted and $\dim(\operatorname{span}[L]_{\operatorname{Deg}_{\overline{\boldsymbol{x}}}}) = 2|L|-1$ has already been proved.

Now for the inductive case. Assume that the thesis holds for $\mathscr{L}_{A_k}$, and given $[e_1|e_2] = \boldsymbol{a}^{k+1}$ take $L_1, L_2 \in \mathscr{L}_{A_k}$ such that $e_1 \in L_1, e_2 \in L_2$. Since $\mathscr{L}_{A_k}$ and $\mathscr{L}_{A_{k+1}}$ are identical for all elements that are not L_1, L_2, their duals and the results of their merging, and since $\operatorname{supp}_{E^n_{\overline{\boldsymbol{x}}}}([e_1|e_2])$ is disjoint from such circuits L' we have that $[L']_{A_{k+1}} = [L']_{A_k}$. It is therefore sufficient to prove the thesis on those $L \in \mathscr{L}'$ resulting from the merging of L_1, L_2 with $[e_1|e_2]$. We study separately the different possibilities for L_1 and L_2.

- ▷ If $L_2 = \overline{L_1}^{A_k}$ then $\mathscr{L}' = \{L_1, \overline{L_1}^{A_k}\}$, but since $e_1 \in L_1, e_2 \in \overline{L_1}^{A_k} = L_2$ Lemma 2.17 implies $\operatorname{span}[L]_{A_{k+1}} = \operatorname{span}[L]_{A_k}$ for all $L \in \mathscr{L}'$, obtaining this way the thesis.

- ▷ If $L_2 = L_1$ and L_1 is not shorted (otherwise also $L_2 = \overline{L_1}^{A_k}$ and it would fall again in the previous case) let $L = L_1 \cup \overline{L_1}^{A_k}$ be the only element of $\mathscr{L}'$, which is shorted in A_{k+1}. Then $[e_1|e_2]$ is independent from $[L]_{A_k}$ (from Lemma 2.17), thus

$$\dim(\operatorname{span}[L]_{A_{k+1}}) = \dim(\operatorname{span}[L_1]_{A_k} \oplus \operatorname{span}\{[e_1|e_2]\}) = (2|L_1| - \cancel{1}) + \cancel{1} = |L|.$$

- ▷ If $L_2 \notin \{L_1, \overline{L_1}^{A_k}\}$ and both L_1 and L_2 are shorted then $L = L_1 \cup L_2$ is again the only element of $\mathscr{L}'$ and is shorted; also since L_1, L_2 belong to different pairs they generate disjoint subspaces $\operatorname{span}[L_1]_{A_k}$ and $\operatorname{span}[L_2]_{A_k}$, therefore

$$\dim(\operatorname{span}[L]_{A_{k+1}}) = \dim\left(\operatorname{span}[L_1]_{A_k} \oplus \operatorname{span}[L_2]_{A_k}\right) = |L_1| + |L_2| = |L|;$$

- ▷ If $L_2 \notin \{L_1, \overline{L_1}^{A_k}\}$ and at least one between L_1 and L_2 is not shorted (say L_2), again $\operatorname{span}[L_1]_{A_k}$ and $\operatorname{span}[L_2]_{A_k}$ are disjoint since L_1, L_2 belong to different pairs. Consider $\sigma^{A_k}_{L_2}$ as defined in the proof of Lemma 2.17: while $[e_1|e_2] \notin \ker \sigma^{A_k}_{L_2}$, it is easy to see that $\operatorname{span}[L_1]_{A_k} \oplus \operatorname{span}[L_2]_{A_k} \subseteq \ker \sigma$ since all its generators lie in $\ker \sigma$. Therefore $[e_1|e_2]$ is not contained in $\operatorname{span}[L_1]_{A_k} \oplus \operatorname{span}[L_2]_{A_k}$. Now,

  - ▷ if L_1 is shorted then $L = L_1 \cup L_2 \cup \overline{L_2}^{A_k}$ is again the only element of $\mathscr{L}'$ and is shorted, and we have

$$\begin{aligned}\dim(\operatorname{span}[L]_{A_{k+1}}) &= \dim\left(\operatorname{span}[L_1]_{A_k} \oplus \operatorname{span}[L_2]_{A_k} \oplus \operatorname{span}\{[e_1|e_2]\}\right)\\ &= |L_1| + (2|L_2| - \cancel{1}) + \cancel{1} = |L|.\end{aligned}$$

  - ▷ if L_1 is also not shorted then $\mathscr{L}' = \{L_1 \cup \overline{L_2}^{A}, \overline{L_1}^{A} \cup L_2\}$ and both elements of $\mathscr{L}'$ are not shorted. For each $L \in \mathscr{L}'$ we therefore have

$$\begin{aligned}\dim(\operatorname{span}[L]_{A_{k+1}}) &= \dim\left(\operatorname{span}[L_1]_{A_k} \oplus \operatorname{span}[L_2]_{A_k} \oplus \operatorname{span}\{[e_1|e_2]\}\right)\\ &= (2|L_1| - \cancel{1}) + (2|L_2| - 1) + 1 = 2|L| - 1.\end{aligned}$$

□

Circuit Extremality Algorithm

Having proved all the required results, we are now able to define the procedure to check for the extremality of a given solution $\overline{\boldsymbol{x}} \in \mathfrak{P}_2^n$, thus proving whether $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$. The algorithm considers the arcs $e \in E^n$ for which $\overline{x}_e = 1/2$ and partitions them in circuits using the set of degree vectors; the partitions are then repeatedly merged using the vectors of active subtour elimination constraints. The algorithm stops whenever the circuit pairs in the partitioning are all shorted, which guarantees the rank maximality of the constraints active at $\overline{\boldsymbol{x}}$ an thus the extremality of the solution. If the subtour elimination constraints are exhausted before the terminating condition is met, the active constraints are not maximal and $\overline{\boldsymbol{x}} \notin \mathfrak{X}_2^n$. Checking for the terminating condition is not achieved in a direct manner, but by counting the times when a partition is shorted by merging it with another shorted partition.

Proposition 2.19 (circuit extremality check)**.** *Given $\overline{\boldsymbol{x}} \in \mathfrak{P}_2^n$ proceed as follows.*

1. *Find $E^n_{\overline{\boldsymbol{x}}} = \{e \in E^n \mid \overline{x}_e = 1/2\}$ and determine*

$$\mathrm{Deg}_{\overline{\boldsymbol{x}}} \coloneqq \left\{ \mathrm{pr}_{\langle E^n_{\overline{\boldsymbol{x}}} \rangle^n} \left(\mathrm{deg}^{\pm,n}_u \right) \;\middle|\; u \in V^n \right\}, \quad \mathrm{Sep}_{\overline{\boldsymbol{x}}} \coloneqq \left\{ \mathrm{pr}_{\langle E^n_{\overline{\boldsymbol{x}}} \rangle^n} \left(\mathrm{sep}^n_S \right) \;\middle|\; S \in \mathscr{S}^n_{\mathtt{ACT}} \left(\overline{\boldsymbol{x}} \right) \right\},$$

   compute $\mathscr{L}_A$ having initialized $A = \mathrm{Deg}_{\overline{\boldsymbol{x}}}$, then set $c = \frac{1}{2} |\mathscr{L}_A|$.

2. *Select $[e_1|e_2] \in \mathrm{Sep}_{\overline{\boldsymbol{x}}} \setminus A$ and add it to A, merging the partitions of $L_1, L_2 \in \mathscr{L}_A$ as in proposition Proposition 2.15: if $L_2 \neq \overline{L_1}$ and at least one between L_1, L_2 is non-shorted, decrease c by 1.*

3. *If c has reached 0, then $\overline{\boldsymbol{x}} \in \mathfrak{X}_2^n$, otherwise repeat point (2.); if all elements of $\mathrm{Sep}_{\overline{\boldsymbol{x}}}$ have been selected, $\overline{\boldsymbol{x}} \notin \mathfrak{X}_2^n$.*

Proof. c effectively counts the number of non-shorted circuit pairs in $\mathscr{L}_A$, since a simple case check shows that the number of non-shorted circuits pairs in $\mathscr{L}_A$ decreases by a unit if and only if the condition in point (2.) is met by L_1, L_2. The terminating condition of $c = 0$ is therefore equal to the request that all circuit pairs be shorted. The fact that circuit pairs generate disjoint subspaces which are full rank if and only if the pair is shorted, together with the fact that it is sufficient to check for the rank maximality of the coefficient-vectors of active constraints over the set of 1/2-arcs of $\overline{\boldsymbol{x}}$ proves the correctness of the algorithm. □

When dealing with the implementation of the algorithm, it turns out to be more efficient to assign to each arc a label corresponding to the circuit the arc belongs to. This allows for a faster retrieval of the partitions L_1, L_2 that contain the extremities of the selected active subtour elimination vector at each iteration. Updating the arc labels at each partitions merging however becomes highly unpractical: therefore it is preferable to leave unchanged the label of the initial partitioning at each arc, separately keeping track of which partitions get merged together and of the duals of each partition. An example implementation of this kind can be found in Algorithm 2.4.

In in Section §3.1, the performance results of an implementation of Algorithm 2.4 are displayed and compared to those of the direct rank-maximality check: the circuit extremality algorithm is shown to provide a many-times improvement in computing speed (around 350× for $n = 11$), with its runtime growing not nearly as rapidly with n as the direct method.

Algorithm 2.4 Circuit extremality algorithm

▷ **Input** A solution $\overline{\boldsymbol{x}} \in \mathfrak{P}_2^n$

▷ **Output** The membership of $\overline{\boldsymbol{x}}$ in $\mathfrak{X}_2^n$

```
c, part ← CircuitPartition(x̄)
duals: (k, i) ↦ {(k, 1 − i)};  shorted ← ∅
for [e1, e2] ∈ Sep_x̄ do
    L1 ← part(e1);  L2 ← part(e2)
    if L2 ∉ duals(L1) ∧ (L1 ∉ shorted ∨ L2 ∉ shorted) then
        c ← c − 1
        if c = 0 then
            return True
        end if
        duals, shorted ← MergePartitions(L1, L2, duals, shorted)
    end if
end for
return False
```

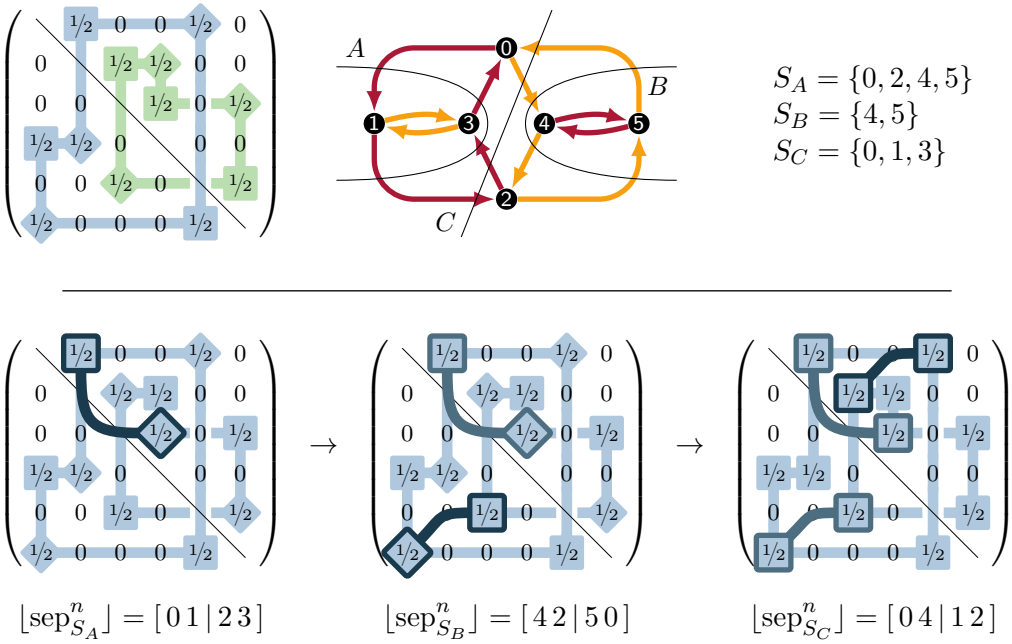


Figure 2.8: Schematic representation of a circuit extremality algorithm example run over the instance $\overline{\boldsymbol{x}}$ of Figure 2.6b; the projections of the active subtour elimination vectors relative to the subsets S_A, S_B, S_C are successively added, and at each step the effects on the circuit partitions are shown similarly to the previous figures: the existence of a single, shorted circuit at the end proves that the input $\overline{\boldsymbol{x}}$ is an extreme point of $\mathfrak{P}^n$.

Algorithm 2.5 Arcs circuit partitioning

1. **function** CIRCUITPARTITION($\overline{\boldsymbol{x}}$)
2. $E^n_{\overline{\boldsymbol{x}}} \leftarrow \{e \in E^n \mid \overline{x}_e = 1/2\}$
3. $c \leftarrow 0$; $E \leftarrow E^n_{\overline{\boldsymbol{x}}}$; $\mathsf{part}\colon E^n_{\overline{\boldsymbol{x}}} \to \mathbb{N} \times \{0,1\}$
4. **repeat**
5. $c \leftarrow c + 1$
6. Select $uv \in E$
7. **repeat**
8. $\mathsf{part}\,(uv) \leftarrow (c,0)$; $E \leftarrow E \setminus \{uv\}$
9. Select $u'v'$ from $E^n_{\overline{\boldsymbol{x}}}$ with $u' = u, v' \neq v$
10. $\mathsf{part}\,(uv) \leftarrow (c,1)$; $E \leftarrow E \setminus \{u'v'\}$
11. Select uv from $E^n_{\overline{\boldsymbol{x}}}$ with $u \neq u', v = v'$
12. **until** $uv \in E$
13. **until** $E = \emptyset$
14. **return** c, part
15. **end function**

Algorithm 2.6 Partitions merging

1. **function** MERGEPARTITIONS($L_1, L_2, \mathsf{duals}, \mathsf{shorted}$)
2. **if** $L_1 \neq L_2 \wedge \{L_1, L_2\} \cap \mathsf{shorted} \neq \emptyset$ **then**
3. Get $L \in \{L_1, L_2\}$ with $L \notin \mathsf{shorted}$
4. $\mathsf{duals}\,(L) \leftarrow \mathsf{duals}\,(L) \cup \{L\}$
5. **end if**
6. **for** $(A, B) \in \{(L_1, L_2), (L_2, L_1)\}$ **do**
7. $\mathsf{duals}\,(A) \leftarrow \mathsf{duals}\,(A) \cup \bigcup_{L \in \mathsf{duals}(B)} \mathsf{duals}\,(L)$
8. **end for**
9. **for** $(A, B) \in \{(L_1, L_2), (L_2, L_1)\}$ **do**
10. **for** $L \in \mathsf{shorted}\,(B)$ **do**
11. $\mathsf{duals}\,(L) \leftarrow \mathsf{duals}\,(A)$
12. **end for**
13. **if** $A \in \mathsf{duals}\,(A)$ **then**
14. $\mathsf{shorted} \leftarrow \mathsf{shorted} \cup \mathsf{duals}\,(A)$
15. **end if**
16. **end for**
17. **return** duals, $\mathsf{shorted}$
18. **end function**

2.4 Generation

Having dealt with the problem of determining when a proposed solution $\overline{\boldsymbol{x}}$ is unique up to isomorphism, is a feasible solution and is an extreme point of $\mathfrak{P}^n$, we now take a step back to discuss the generation of such instances $\overline{\boldsymbol{x}}$. As discussed, instance are generated as a weighted sum of cycle covers of $\mathcal{K}_n$ in vector form $\overline{\boldsymbol{x}} = \frac{1}{\alpha}\sum_{i=1}^{\alpha}\overline{\boldsymbol{y}}^i$, and to deal with the cover-sets $\left\{\overline{\boldsymbol{y}}^1, \dots, \overline{\boldsymbol{y}}^\alpha\right\} \subset \mathfrak{C}^n$ in a combinatorial way we leverage the encoding developed in Section §2.2. Therefore the task is now to generate all possible standard form encodings of cover-sets, focusing on the case $\alpha = 2$.

Half-Integer Generator

The generation of the two cycles $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2$ is achieved by iterating first over the set $\mathscr{P}_n$ of all possible integer partitions of n (each element with its components in decreasing order) taking $P^1, P^2 \in \mathscr{P}_n$ with $P^i = \left(p_1^i, \dots, p_{N_i}^i\right)\ \forall\, i \in \{1, 2\}$ and $P^1 \geq_{\text{lex}} P^2$ as the partitions of the encodings ξ_1, ξ_2 of $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2$ respectively.

Remark 2.20. We also ask for $n \geq p_j^1 \geq 2$ and $n - 2 \geq p_j^2 \geq 2$ for all $j \in \{1, ..., N_i\}$, the lower bound coming in both cases from the definition of cover encoding, the upper bound on p_j^2 instead from the fact that otherwise $P^1 = P^2 = (n)$, thus making both $\overline{\boldsymbol{y}}^1$ and $\overline{\boldsymbol{y}}^2$ tours, which in turn implies that $\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2$ as well as $\overline{\boldsymbol{x}}$ are feasible ($\overline{\boldsymbol{y}}^1, \overline{\boldsymbol{y}}^2, \overline{\boldsymbol{x}} \in \mathfrak{P}^n$); in case $\overline{\boldsymbol{y}}^1 = \overline{\boldsymbol{y}}^2$ then $\overline{\boldsymbol{x}} = \overline{\boldsymbol{y}}^1 \in \mathfrak{C}^n$ and therefore $\textsc{gap}\left(\overline{\boldsymbol{x}}\right) = 1$ making it uninteresting at the purpose of calculating Gap_n; otherwise if $\overline{\boldsymbol{y}}^1 \neq \overline{\boldsymbol{y}}^2$ we would have $\overline{\boldsymbol{x}} \notin \mathfrak{X}^n$, since linear programming theory shows that no extreme solution can be the convex combination of two distinct feasible solutions.

For each $P^1, P^2 \in \mathscr{P}_n$ then from the requirement that (ξ_1, ξ_2) be a standard form encoding the first cover encoding ξ_1 is uniquely determined as in equation (2.1). All the possibilities for the second cover encoding ξ_2 are instead generated in a recursive fashion by the function specified in Algorithm 2.8, which relies on the routines $\textsc{Combinations}\,(V, p)$ returning all the combinations of p elements of V (which are assumed to be sorted in ascending order) and $\textsc{Permutations}\,(U)$ that returns all possible permutations of the list U. The complete generating procedure of the encoding of 2-covers sets can be found in Algorithm 2.7.

Algorithm 2.7 Cover-set encodings generation

▷ **Input** A natural number n
▷ **Output** A set Ξ of the encodings of all 2-cover sets on $\mathcal{K}_n$
1. $V^n \leftarrow \{0, \dots, n-1\}\,;\ \Xi \leftarrow \emptyset$
2. int_part $\leftarrow \textsc{IntegerPartitions}\,(n)$ ▷ *each partition with* $2 \leq p_j \leq n$
3. **for** $P^1 = \left(p_1^1, \dots, p_{N_i}^1\right) \in$ int_part **do**
4. $\quad \xi_1 \leftarrow \left[\, 0 \dots (p_1^1 - 1) \,\middle|\, \cdots \,\middle|\, (n - p_{N_1}^1 - 1) \dots (n-1) \,\right]$
5. $\quad$ **for** $P^2 \in$ int_part **with** $P^2 \leq_{\text{lex}} P^1$ and $2 \leq p_j^2 \leq n-2$ **do**
6. $\quad\quad$ **for** $\xi_2 \in \textsc{HalfIntegerCovers}\left(V^n, P^2, 0, 0\right)$ **do**
7. $\quad\quad\quad \Xi \leftarrow \Xi \cup \{(\xi_1, \xi_2)\}$
8. $\quad\quad$ **end for**
9. $\quad$ **end for**
10. **end for**

Algorithm 2.8 Half-integer covers generation

```
1.  function HalfIntegerCovers(V, P, last_part, last_top)
2.      if P = ∅ then
3.          return ∅
4.      end if
5.      C ← ∅;  part ← P[0]
6.      for head ∈ Combinations(V, p) do          ▷ head sorted in ascending order
7.          top ← head[0]
8.          if part = last_part ∧ top < last_top then
9.              continue
10.         end if
11.         V' ← V \ head
12.         for perm ∈ Permutations(head[1:]) do
13.             ξ' ← [ top perm[0] ... perm[part − 2] ]
14.             for tail ∈ HalfIntegerCovers(V', P[1:], part, top) do
15.                 ξ ← ξ' + tail
16.                 C ← C ∪ {ξ}
17.             end for
18.         end for
19.     end for
20.     return C
21. end function
```

Pure Half-Integer Generator

As showed in §1.2.3, when estimating Gap_n it is sufficient to limit the search to instances $\overline{\boldsymbol{x}}$ that have no arcs e with $\overline{x}_e = 1$. As such, the efficiency of our search is increased if the generating procedure avoids entirely the non-pure half-integer solutions. This can be obtained by slightly adjusting the existing procedures: Algorithm 2.8 can be modified by introducing a function $\mathsf{succ}_{P^1} \colon V^n \to V^n$ that maps each vertex $u \in V^n$ to its successor in the cycle cover $\overline{\boldsymbol{y}}^1$, that is for all $u \in V^n$

$$\underset{P^1}{\mathsf{succ}} \colon u \longmapsto S^{P^1}_{m-1} + \left[v - S^{P^1}_{m-1} + 1 \mod p^1_m \right]$$

with $S^{P^1}_m \coloneqq \sum_{j=1}^{m} p^1_j$ (let $S^{P^1}_0 \coloneqq 0$) and $m \in \{1, ..., N_1\}$ such that $S^{P^1}_{m-1} \leq u < S^{P^1}_m$. Whenever the vertex u has the same successor v in both $\overline{\boldsymbol{y}}^1$ and $\overline{\boldsymbol{y}}^2$ then the resulting $\overline{\boldsymbol{x}}$ must have $\overline{x}_{uv} = 1$ (the converse also being true) and can therefore be safely discarded. Algorithm 2.9 implements the described check, and the relative generating procedure can be obtained from Algorithm 2.7 by switching the call to the function HalfIntegerCovers with that to the function PureHalfIntegerCovers.

Algorithm 2.9 Pure half-integer covers generation

```
1.  function PureHalfIntegerCovers(V, P, last_part, last_top)
    <Algorithm 2.8 lines 2-13>
2.      is_half ← True
3.      for i ∈ {0, part − 1} do
4.          if succ_{P^1}(ξ'[i]) = ξ'[(i + 1) mod part] then
5.              is_half ← False
6.          end if
7.      end for
8.      if is_half then
9.          for tail ∈ PureHalfIntegerCovers(V', P[1 :], part, top) do
10.             ξ ← ξ' + tail
11.             C ← C ∪ {ξ}
12.         end for
13.     end if
    <Algorithm 2.8: lines 18-20>
14. end function
```

3. Computational Results

The procedure described in the last chapter has been implemented in software as part of this work. The code handling the generation of the instances, the database storage of the results and the parallelization of the computation, has been written in Python programming language; the isomorphism-class check relies on the C-based library NAUTY [23] via the Python package pynauty [11], while the remaining property-checking routines are implemented directly in Python; the state-of-the-art solver Gurobi [18] with its Python interface gurobipy has been used for solving the LP problem GAP for each selected instance. The software ran on a Dell Precision 7960 Tower Workstation with a 112-threads Intel Xeon w9-3495X CPU and 130 GiB of memory, completing its run up to $n = 11$ and providing partial results for $n = 12$. The obtained results considerably strengthen the results of [13], replicating them in a much shorter runtime for $n \leq 9$ and extending them to the complete set of half-integers for n up to 11.

All the code produced for this work is available at [30].

3.1 Implementation Details

Following the construction of Chapter 2, the implementation is structured as follows:

- ▷ the encodings of candidates $\overline{\boldsymbol{x}}$ are generated as in Section §2.4 and saved to a database;
- ▷ all $\overline{\boldsymbol{x}}$ satisfying the properties of Section §2.3 are selected and the database is updated;
- ▷ the integrality gap of the selected $\overline{\boldsymbol{x}}$ is computed and again inserted into the database.

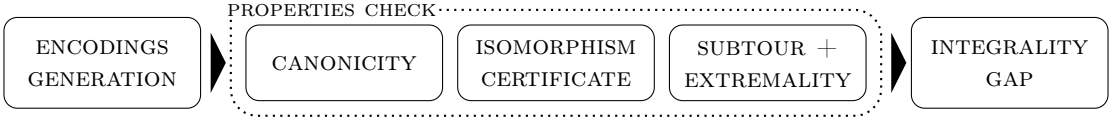


Figure 3.1: Structure of the software implementation.

The current section provides a quick overview of the details of each part of the software.

Generation Generating the cover-set encodings of all half-integer solutions has been achieved by implementing Algorithm 2.7 in Python with both cover-generation Algorithms 2.8 and 2.9 presented in Section §2.4. To improve performance on the high number of recursive calls, a result-caching system has been added with the Python module functools to all functions IntegerPartitions, HalfIntegerCovers, and PureHalfIntegerCovers. Since it is sufficient to compute the integrality gap on

the pure half-integers, only the function PureHalfIntegerCovers will be generally used, as it decreases the number of generated instances by about 65-70% (see Figure 3.2).

n	half-integer	pure half-int.	
4	6	3	-50.0%
5	40	11	-72.5%
6	360	112	-68.9%
7	3108	978	-68.5%
8	33579	10809	-67.8%
9	356816	116463	-67.4%
10	4621716	1527602	-66.9%
11	59724500	19938367	-66.6%

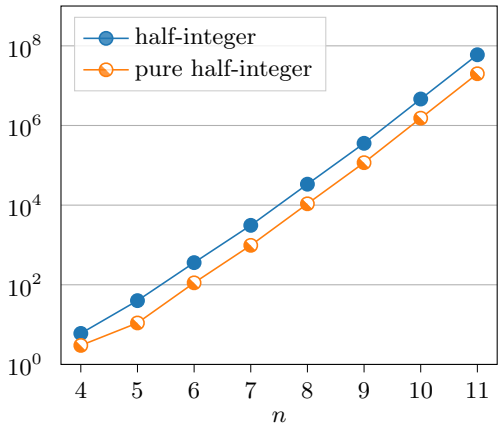


Figure 3.2: Number of generated instances: half-integer vs pure half-integer.

Database All generated solutions are saved to a SQLite database, which is successively updated with the results of the computations. To access the database from Python the SQLAlchemy library has been used, providing an Object Relational Mapper on top of its SQL toolkit. The database also keeps track of the runtime of each operation on every instance for performance analysis.

Parallelization To reduce runtime all computations of properties and of the integrality gap have been duly parallelized through the standard Python multiprocessing library. The main program thread handles the distribution of batches of instances to the child processes, the retrieval of the computed results, and the periodical committing of the results to the database.

Property checking strategy Before proceeding with the performance analysis of the property checking procedures, Figure 3.3 shows how the circuit extremality algorithm

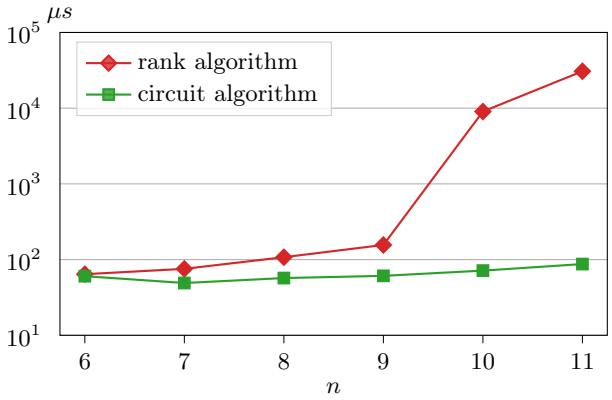


Figure 3.3: Mean runtime over 10^4 sample pure half-integer instances* of the circuit extremality algorithm compared to the active constraints rank check algorithm.

(Algorithm 2.4) developed in Section §2.3 performs a lot better than the direct rank-maximality check of the active constraints vectors, growing much more slowly in runtimes with n, effectively justifying its choice for the subtour elimination and extremality checking algorithm.

The three properties — canonicity, isomorphism class (through the NAUTY certificate), and subtour elimination and extremality — are successively checked on all generated solutions. Since the ultimate goal is to select only instances that satisfy all properties, as soon as one property is not satisfied there is evidently no need to check for the remaining ones. When talking about the isomorphism class certificate, with "satisfaction" we intend for simplicity the selection of one instance per each unique certificate (usually the first encountered) that will be the one "satisfying" the isomorphism class property, while the discarded instances do not. Recall at this point how checking for canonicity after having checked the isomorphism certificate is unnecessary, as translating an encoding in canonic form does not change the isomorphism class.

A selection of the optimal sequence of property checks (we refer to this as a *strategy*), based on the computation time of each checking procedure and on the extent of the reduction in the number of instances that are passed to the next step, is thus required in order to reduce overall computation time. Table 3.1 breaks down the number of pure half-integer instances satisfying each of property, while Figure 3.4 shows the average computation times for all three checking procedures described in Section §2.3.

SUB+EXT	CANON	CERT	n = 4	5	6	7	8	9	10	11
			3	11	112	978	10809	116463	1527602	19938367
✓			1	11	71	715	6855	71505	881427	10892612
	✓		2	2	24	102	860	7433	75923	847325
	✓	✓	2	2	21	73	607	4539	46394	466298
✓	✓		1	2	14	70	531	4368	41446	429931
✓	✓	✓	1	2	11	52	365	2931	26906	274134

Table 3.1: Number of pure half-integer instances when successively imposing each property.

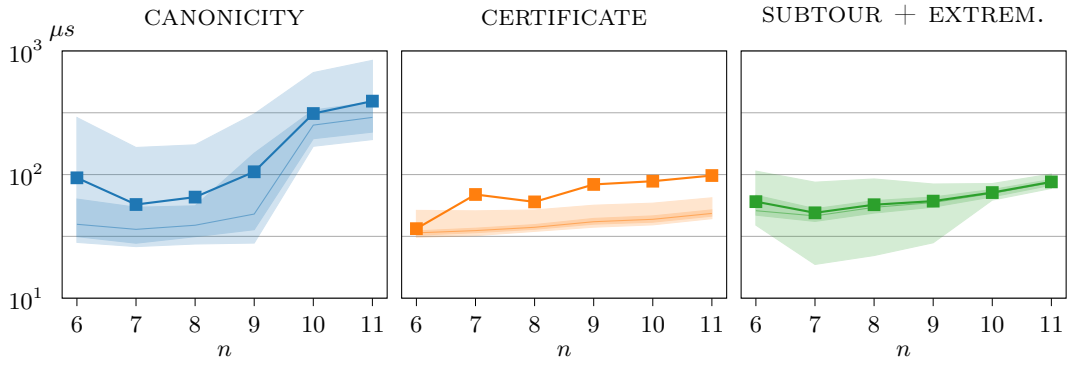


Figure 3.4: Mean runtime over 10^4 sample pure half-integer instances* of the property checking algorithms; the 5-95th (lighter) and 25-75th (darker) percentile regions and the median runtime are also plotted for each property check.

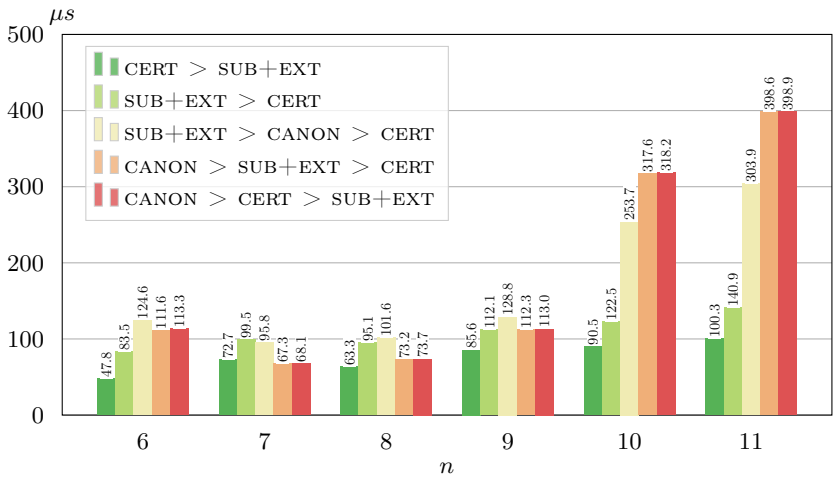


Figure 3.5: Expected runtimes of the property checking procedures by strategy.

This data allows us to estimate the expected mean runtime of the property check of a given instance, the results of which can be seen in Figure 3.5. It can be observed how skipping the encoding canonicity check entirely is generally the quickest choice, due to the high computational cost of the procedure and its redundancy when the isomorphism class is computed. Despite the comparable average runtimes, it is also best to compute the isomorphism class before checking for the subtour elimination constraints and the extremality due to the greater number of instances discarded in the process. The best strategy is thus to check first for the isomorphism certificate and then for the subtour elimination and extremality of the candidate solution.

Computation of the integrality gap To compute $\textsc{gap}(\overline{\boldsymbol{x}})$ for each instance $\overline{\boldsymbol{x}} \in \mathfrak{H}_2^n$ satisfying the three properties the GAP problem has been implemented with the gurobipy Python library and repeatedly solved with Gurobi. By optimizing the gurobipy implementation and the Gurobi solver hyperparameters, performance can be improved in this step with regards in particular to computation time and memory consumption, the last one being the main bottleneck to scaling and parallelization.

With regards to the gurobipy implementation two aspects have been considered: first on-the-fly generation of the GAP problem model (1.8) at each new process initialization is compared with the loading from a file in .mps format of the previously generated model; then the implementation of an early stopping procedure that interrupts the computation whenever the best known lower bound on the value of $\textsc{gap}(\overline{\boldsymbol{x}})$ is above a given threshold (usually Gap_{n-1}^{-1}) is tested. The results of this comparison are displayed in Figure 3.6, which shows together with the ensuing figures the performance of different parameters and methods over 100 sample instances with $n = 11$. It can be seen how model loading performs better than on-the-fly generation (the not-shown process startup time also being significantly quicker), while the similar time performance of the early-stopping procedure is not met by an improvement in memory consumption, which also greatly increases in variance — possibly due to the high number of callback calls to the Python routine used to implement the early-stopping.

*only 68 instances for $n = 6$, 628 for $n = 7$ and 7696 for $n = 8$.

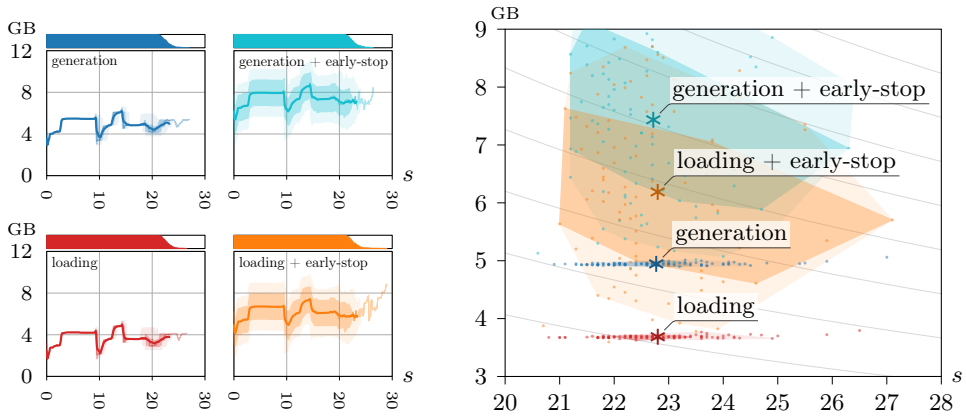


Figure 3.6: Time-memory performances of the different gurobipy GAP implementations; on the left, plots of the average instantaneous memory consumption over time for each configuration with aligned on top the percentage of solving processes still running at each moment; on the right, a scatter plot of the instances by solve time and mean memory consumption, and their global average for each configuration; in every case the 5-95th (lighter) and 25-75th (darker) percentiles regions are shown, while the grid in the right plot is made of hyperbolas of equal time × memory.

Figure 3.7 shows instead the performances of the different Gurobi solving algorithm ("Method" parameter of the solver): the GAP problem is most efficiently solved by the dual simplex algorithm, the barrier and primal algorithm performing worse time- and memory-wise and the concurrent one having a similar time performance (since it includes the dual algorithm) but significantly higher memory usage (due to its parallel nature).

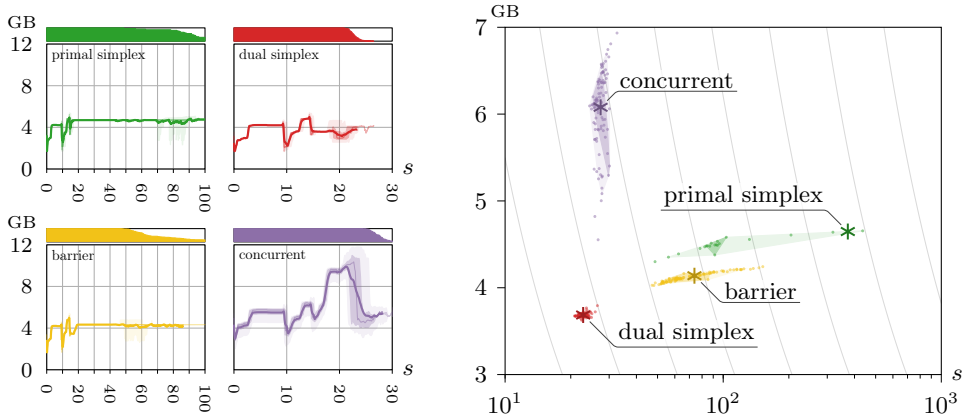


Figure 3.7: Time-memory performances of the different Gurobi solving algorithms.

Finally, focusing on the Gurobi solver "Presolve" and "PreSparsify" parameters, for which the options are *off* (0), *conservative* (1), or *aggressive* (2) for "Presolve", and *off* (0) or *on* (2) for "PreSparsify", we can observe how the most efficient combination is a conservative "Presolve" and no PreSparsify, as illustrated by Figure 3.8.

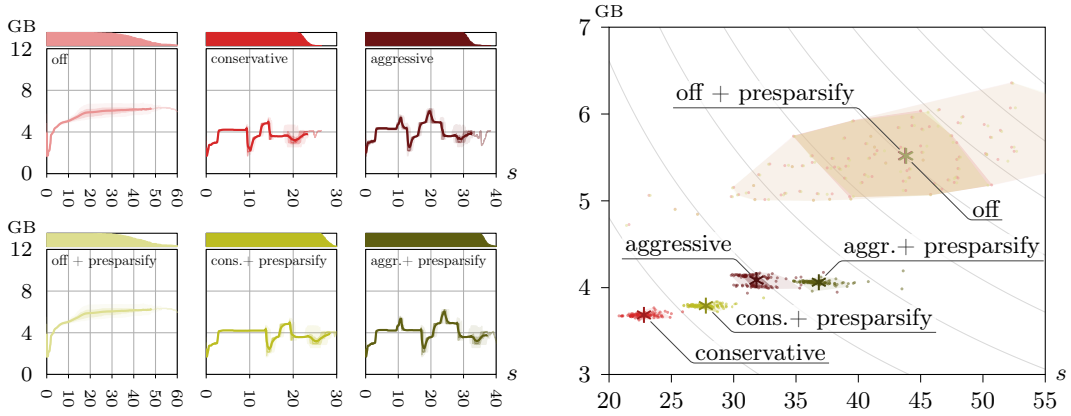
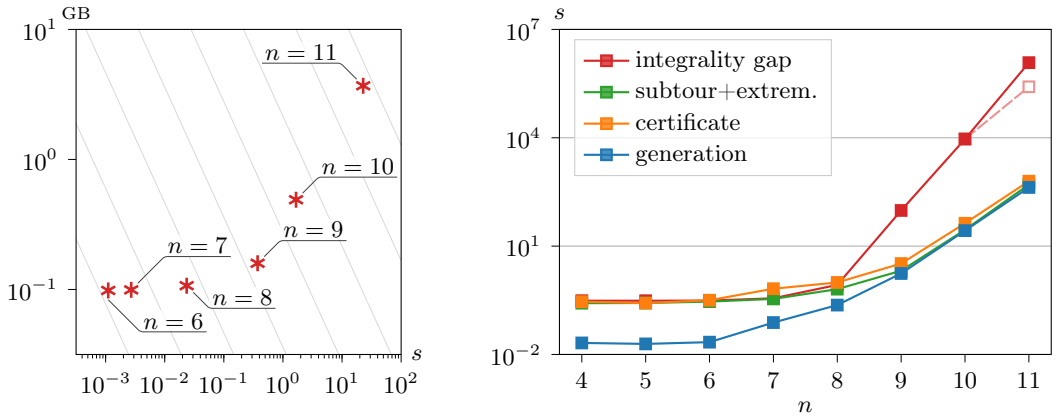


Figure 3.8: Time-memory performances of the different Gurobi parameter combinations.

Although the remarks and optimization expedients presented up to now help in substantially reducing the resources needed by the computation, the time and memory requirements still grow exponentially with n. For $n = 11$ already our software had to be restrained to using only 24 out of the 112 threads available due to the "limited" (130 GiB) memory of our machine, and ran for almost two weeks; for $n = 12$ the number of threads had to be further reduced to 2 — each thread taking up almost 60 GB of memory — and the exhaustive computation of all instances proved to be unfeasible in a reasonable amount of time, the best runtime estimates being in the range of a few years.

To conclude, Figure 3.9 summarizes the final performance of the software, displaying for each n the time and memory required to compute the integrality gap of each instance (3.9a), and the total runtimes of each step in the procedure, from instance generation to gap computation (3.9b).



(a) Mean time-memory requirements by n for the integrality gap computation.

(b) Total runtime by n of each computation step (for $n = 11$ the integrality gap was computed using only 24 threads; an estimation of the runtime on 112 threads is shown dashed in a lighter color).

Figure 3.9: Breakdown by n of software time-memory performance and total runtime.

3.2 Results

Despite being limited at relatively low values of n by the available computational resources, the software still managed to complete its runs for n up to 11, therefore finding exhaustively for these n the best integrality gaps over the set of pure half-integer solutions (and therefore also over all half-integer solutions). The best integrality gaps achieved are listed in the following Table 3.2.

n	Gap_n	
3	1	1
4	6/5	1.2
5	5/4	1.25
6	4/3	1.33333
7	4/3	1.33333
8	4/3	1.33333
9	11/8	1.375
10	7/5	1.4
11	10/7	1.42857

Table 3.2: Highest integrality gaps found over half-integer solutions ($n = 3$ has only trivial, non-pure half-integer extreme points and is included for completeness).

In their PhD thesis [13], Elliott-Magwood was able to compute the exact value of Gap_n for $n \leq 7$ by generating all vertices of the ASEP polytope: for $n = 3, 4, 5, 6$ this was achieved via direct generation of the vertices with the software packages PORTA [8] and cddr+ [16]; for $n = 7$ the direct generating procedure did not complete after running for more than a week, but by leveraging the solution of different subproblems the author was still able to enumerate all non-isomorphic vertices in over 20 hours of computation. For $n = 8, 9$ the author managed to generate and compute the gap of all half-integer instances up to isomorphism by considering all possible support digraphs of half-integer points and generating in sequence all extreme points with the given digraphs as support. Unfortunately the runtimes of this procedures have not been provided for comparison. For $n \geq 10$ this approach too could not be carried out, and the author restricted his search to only a family of half-integers built by recursively inserting new nodes and arcs with custom-defined "2-jack" operations to half-integer instances of lower dimension n.

The results we obtained are consistent with and strengthen those of Elliott-Magwood for two main reasons:

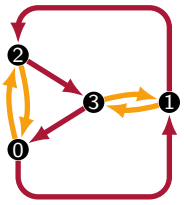
- ▷ firstly, having reproduced them in a fraction of the time, since for $n = 7$ our software run in less than 1.5 seconds (the only n with a runtime provided in [13]) and in less than 2 minutes for $n = 9$ (the largest exhaustive computation in [13]);
- ▷ secondly, proving for $n = 10$ and $n = 11$ that its estimates of the integrality gap are in fact the highest gaps attainable over half-integer instances.

The solutions with best integrality gaps along with other high-gap solutions are showcased in the next section. As stated earlier, for $n = 12$ the software could not complete its run in reasonable time, but in a partial run was still able to find a few instances with high integrality gap; such instances can again be found in Section §3.3.

3.3 High Integrality Gap Instances

$n = 4$

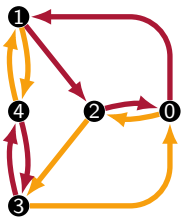
gap: 1.2



$\left[\begin{smallmatrix}0&1&2&3\\0&2&\vert&1&3\end{smallmatrix}\right]$

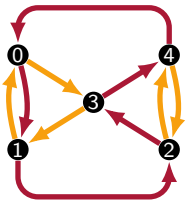
$n = 5$

gap: 1.25



$\left[\begin{smallmatrix}0&1&2&\vert&3&4\\0&2&3&\vert&1&4\end{smallmatrix}\right]$

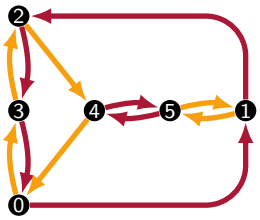
gap: 1.2



$\left[\begin{smallmatrix}0&1&2&3&4\\0&3&1&\vert&2&4\end{smallmatrix}\right]$

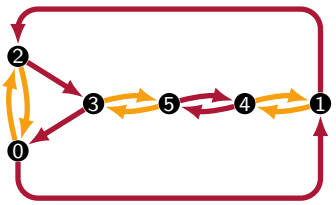
$n = 6$

gap: 1.33333



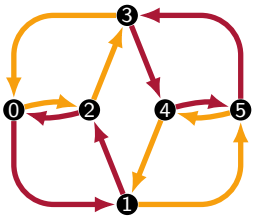
$\left[\begin{smallmatrix}0&1&2&3&\vert&4&5\\0&3&2&4&\vert&1&5\end{smallmatrix}\right]$

gap: 1.27273

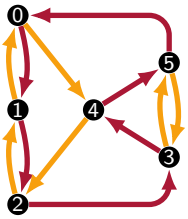


$\left[\begin{smallmatrix}0&1&2&3&\vert&4&5\\0&2&1&4&\vert&3&5\end{smallmatrix}\right]$

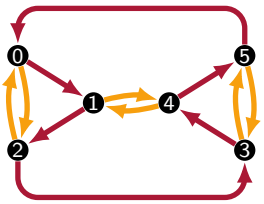
gap: 1.25



$\left[\begin{smallmatrix}0&1&2&\vert&3&4&5\\0&2&3&\vert&1&5&4\end{smallmatrix}\right]$



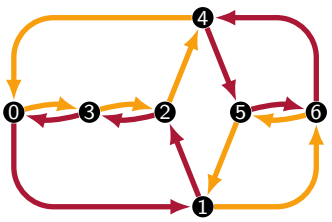
$\left[\begin{smallmatrix}0&1&2&3&4&5\\0&4&2&1&\vert&3&5\end{smallmatrix}\right]$



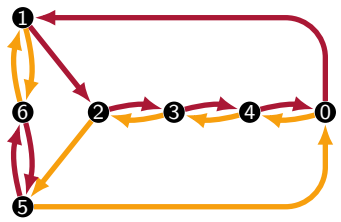
$\left[\begin{smallmatrix}0&1&2&3&4&5\\0&2&1&4&\vert&3&5\end{smallmatrix}\right]$

$n = 7$

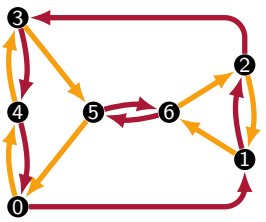
gap: 1.33333



$\left[\begin{smallmatrix}0&1&2&3&\vert&4&5&6\\0&3&2&4&\vert&1&6&5\end{smallmatrix}\right]$



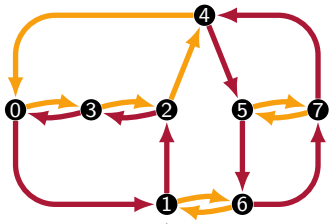
$\left[\begin{smallmatrix}0&1&2&3&4&\vert&5&6\\0&4&3&2&5&\vert&1&6\end{smallmatrix}\right]$



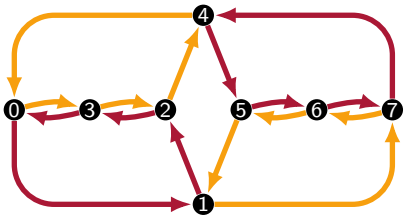
$\left[\begin{smallmatrix}0&1&2&3&4&\vert&5&6\\0&4&3&5&6&\vert&2&1\end{smallmatrix}\right]$

$n = 8$

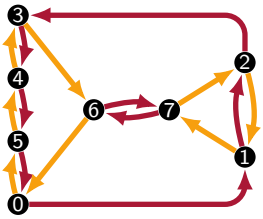
gap: 1.33333



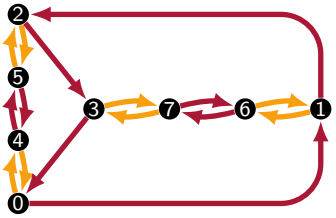
[0123|4567]
[0324|16|57]



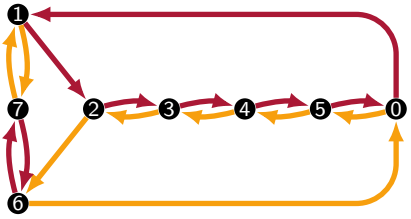
[0123|4567]
[0324|1765]



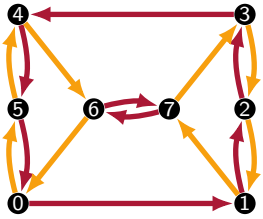
[012345|67]
[05436|172]



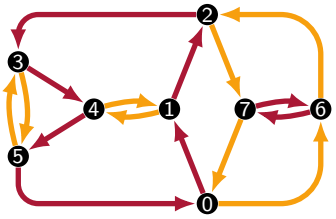
[0123|45|67]
[04|16|25|37]



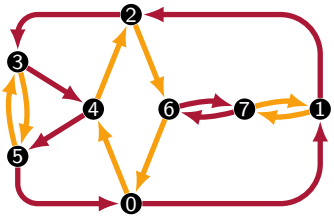
[012345|67]
[054326|17]



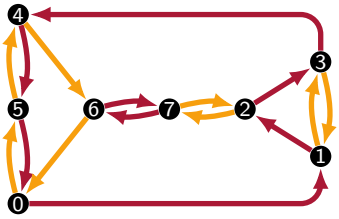
[012345|67]
[0546|1732]



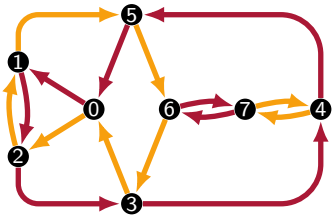
[012345|67]
[0627|14|35]



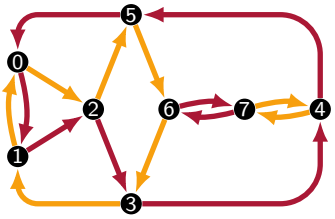
[012345|67]
[0426|17|35]



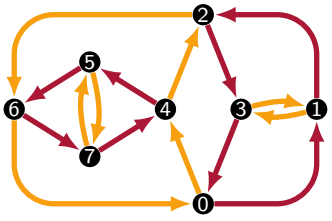
[012345|67]
[0546|13|27]



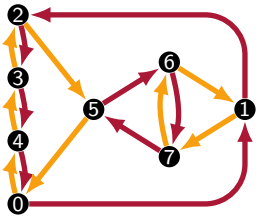
[012345|67]
[021563|47]



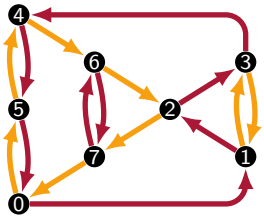
[012345|67]
[025631|47]



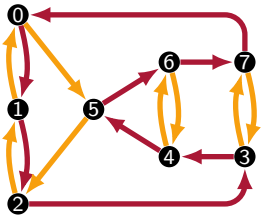
[0123|4567]
[0426|13|57]



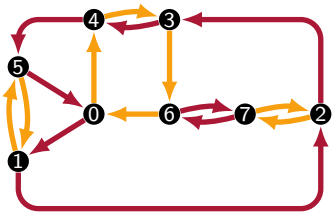
[01234|567]
[04325|176]



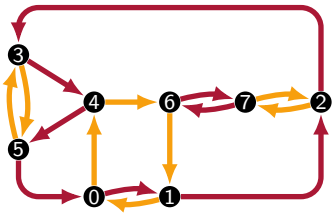
[012345|67]
[054627|13]



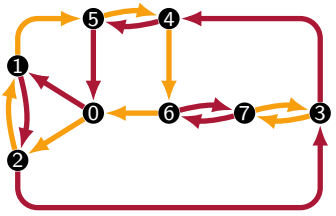
[01234567]
[0521|37|46]



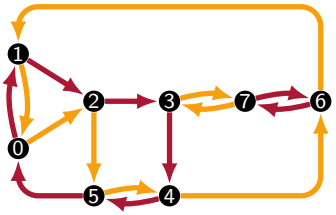
$\left[\begin{smallmatrix}0&1&2&3&4&5&|&6&7\\0&4&3&6&|&1&5&|&2&7\end{smallmatrix}\right]$



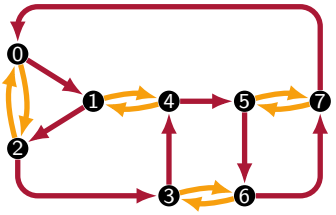
$\left[\begin{smallmatrix}0&1&2&3&4&5&|&6&7\\0&4&6&1&|&2&7&|&3&5\end{smallmatrix}\right]$



$\left[\begin{smallmatrix}0&1&2&3&4&5&|&6&7\\0&2&1&5&4&6&|&3&7\end{smallmatrix}\right]$



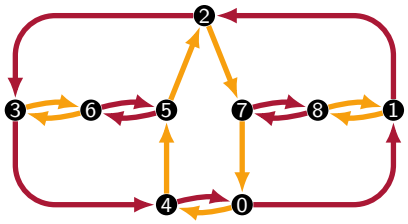
$\left[\begin{smallmatrix}0&1&2&3&4&5&|&6&7\\0&2&5&4&6&1&|&3&7\end{smallmatrix}\right]$



$\left[\begin{smallmatrix}0&1&2&3&4&5&6&7\\0&2&1&4&3&6&|&5&7\end{smallmatrix}\right]$

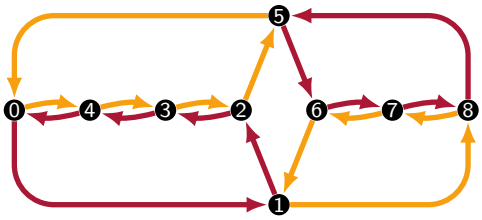
$n = 9$

gap: 1.375



$\left[\begin{smallmatrix}0&1&2&3&4&|&5&6&|&7&8\\0&4&5&2&7&|&1&8&|&3&6\end{smallmatrix}\right]$

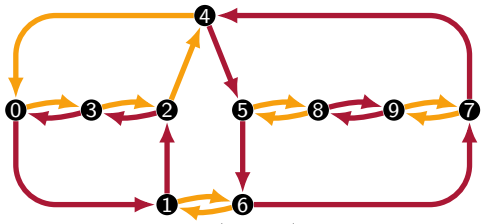
gap: 1.36364



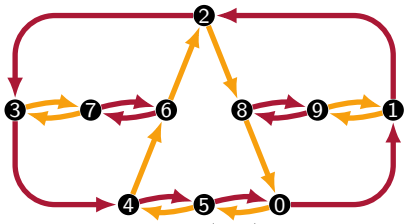
$\left[\begin{smallmatrix}0&1&2&3&4&|&5&6&7&8\\0&4&3&2&1&|&6&7&8&5\end{smallmatrix}\right]$

$n = 10$

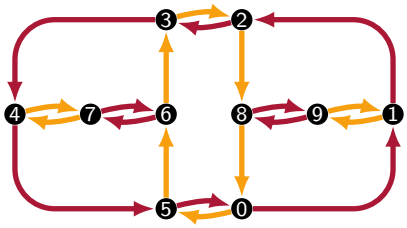
gap: 1.4



$\left[\begin{smallmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&5&8&9&7&|&1&6\end{smallmatrix}\right]$



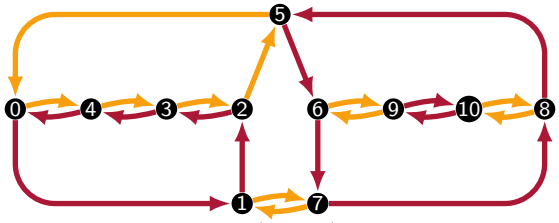
$\left[\begin{smallmatrix}0&1&2&3&4&5&6&7&8&9\\0&5&4&6&2&8&9&1&|&3&7\end{smallmatrix}\right]$



$\left[\begin{smallmatrix}0&1&2&3&4&5&6&7&8&9\\0&5&6&3&2&8&9&1&|&4&7\end{smallmatrix}\right]$

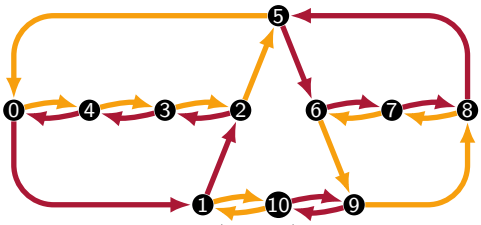
$n = 11$

gap: 1.42857

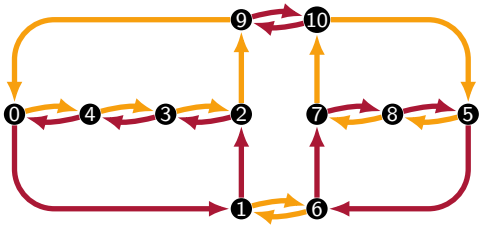


$\left[\begin{array}{c|c|c|c} 0\,1\,2\,3\,4 & 5\,6\,7\,8 & 9\,10 \\ 0\,4\,3\,2\,5 & 1\,7 & 6\,9 & 8\,10 \end{array}\right]$

gap: 1.41176

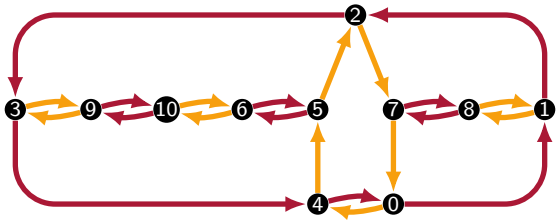


$\left[\begin{array}{c|c|c|c} 0\,1\,2\,3\,4 & 5\,6\,7\,8 & 9\,10 \\ 0\,4\,3\,2\,5 & 6\,9\,8\,7 & 1\,10 \end{array}\right]$



$\left[\begin{array}{c|c|c|c} 0\,1\,2\,3\,4 & 5\,6\,7\,8 & 9\,10 \\ 0\,4\,3\,2\,9 & 5\,8\,7\,10 & 1\,6 \end{array}\right]$

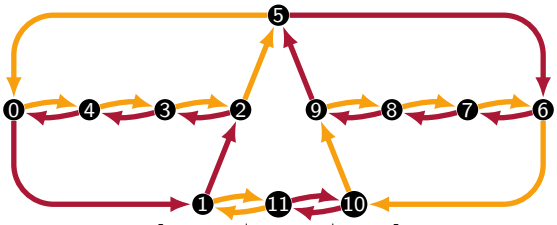
gap: 1.40909



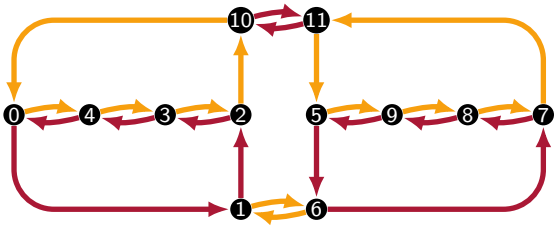
$\left[\begin{array}{c|c|c|c} 0\,1\,2\,3\,4 & 5\,6 & 7\,8 & 9\,10 \\ 0\,4\,5\,2\,7 & 1\,8 & 3\,9 & 6\,10 \end{array}\right]$

$n = 12$

gap: 1.42857



$\left[\begin{array}{c|c|c|c} 0\,1\,2\,3\,4 & 5\,6\,7\,8\,9 & 10\,11 \\ 0\,4\,3\,2\,5 & 6\,10\,9\,8\,7 & 1\,11 \end{array}\right]$



$\left[\begin{array}{c|c|c|c} 0\,1\,2\,3\,4 & 5\,6\,7\,8\,9 & 10\,11 \\ 0\,4\,3\,2\,10 & 5\,9\,8\,7\,11 & 1\,6 \end{array}\right]$

Conclusions

The results shown up to this point seem to suggest that the exhibited approach is a promising one. The significant reduction in the number of explored instances and the high gain in computation speed already allowed us to improve the best known bounds on the ATSP integrality gap, and may hopefully lead to further results in the future. To this last purpose we now present some natural questions and directions of research that arose from the development of the technique and that may serve as basis for prospective improvements.

First of all, any additional optimization of the software could allow for the extension of the computation, potentially making it feasible for $n = 12$ or greater. In this regard, two aspects of the implementation appear to be the best candidates, as there is extensive space for their improvement and they would both provide a great increase in efficiency. Immediately, it is easy to see that any reduction in the number of candidate solutions generated would translate in an equal reduction in computation time; hence, finding any mathematical guarantee on the structure of the solutions with highest gap (such as the ability to remove non-pure half-integer solutions) could be applied directly to our process in order to discard uninteresting instances and enhance performance this way. Otherwise, tackling the main computational bottleneck — memory consumption — would also be very beneficial, as it would make it possible to take advantage of the parallelization potential of the procedure. To this end, we see that the largest portion of memory (volatile and non-volatile) in solving GAP is taken up by the $\text{cost}\,(T) \geq 1$ constraints, which are indexed over all tours T in the complete graph $\mathcal{K}_n$ and thus grow rapidly in number with n. It would therefore be desirable to reduce the memory requirements of this family of constraints, for example by removing them from the original problem and dynamically reintroducing the ones that are violated in the solving process.

Finally, the next straightforward step would be adapting the described approach to the symmetric TSP. The technique in this case has the potential to be very fruitful, as not only it would allow for the exact computation of the integrality gap over half-integer solutions for small n, but it could be helpful in the search for a counterexample to the $\frac{4}{3}$ upper bound conjecture, as it would provide a new perspective on the generation of high-gap STSP instances.

List of Algorithms

List of Tables

List of Figures

Bibliography

[1] Bagchi, T. P., Gupta, J. N. D., and Sriskandarajah, C. A review of TSP based approaches for flowshop scheduling. *European Journal of Operational Research 169*, 3 (Mar. 2006), 816–854.

[2] Benoit, G., and Boyd, S. Finding the Exact Integrality Gap for Small Traveling Salesman Problems. *Mathematics of Operations Research 33*, 4 (Nov. 2008), 921–931.

[3] Birkhoff, G. D. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán, Ser. A 5* (1946), 147–151.

[4] Boyd, S., and Elliott-Magwood, P. Computing the integrality gap of the asymmetric travelling salesman problem. *Electronic Notes in Discrete Mathematics 19* (June 2005), 241–247.

[5] Boyd, S., and Labonté, G. Finding the Exact Integrality Gap for Small Traveling Salesman Problems. In *Integer Programming and Combinatorial Optimization: 9th International IPCO Conference* (Cambridge, MA, USA, May 2002), W. J. Cook and A. S. Schulz, Eds., vol. 2337 of *Lecture Notes in Computer Science*, Springer, pp. 83–92.

[6] Carr, R., and Vempala, S. On the Held-Karp relaxation for the asymmetric and symmetric traveling salesman problems. *Mathematical Programming 100*, 3 (July 2004), 569–587.

[7] Charikar, M., Goemans, M. X., and Karloff, H. J. On the Integrality Ratio for Asymmetric TSP. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science* (USA, Oct. 2004), FOCS '04, IEEE Computer Society, pp. 101–107.

[8] Christof, T., and Löbel, A. POlyhedron Representation Transformation Algorithm (PORTA). version 1.4.0, 1997.
`https://porta.zib.de/`.

[9] Chvátal, V. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics 4*, 4 (Apr. 1973), 305–337.

[10] Dantzig, G., Fulkerson, R., and Johnson, S. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America 2*, 4 (Nov. 1954), 393–410.

[11] Dobsan, P. Pynauty. Github repository, Nov. 2022.
`https://github.com/pdobsan/pynauty`.

[12] Duman, E., and Or, I. Precedence constrained TSP arising in printed circuit board assembly. *International Journal of Production Research 42*, 1 (Jan. 2004), 67–78.

[13] Elliott-Magwood, P. *The Integrality Gap of the Asymmetric Travelling Salesman Problem*. PhD thesis, University of Ottawa, Canada, 2008.

[14] Frieze, A. M., Galbiati, G., and Maffioli, F. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks 12*, 1 (Mar. 1982), 23–39.

[15] Fujimura, K., Obu-Cann, K., and Tokutaka, H. Optimization of surface component mounting on the printed circuit board using SOM-TSP method. In *ICONIP'99. ANZIIS'99 & ANNES'99 & ACNN'99. 6th International Conference on Neural Information Processing. Proceedings (Cat. No.99EX378)* (Nov. 1999), vol. 1, pp. 131–136 vol.1.

[16] Fukuda, K. Cddr+. version 0.76, 1999.
`https://www.cs.mcgill.ca/~fukuda/soft/cddman/cddman.html`.

[17] Gomory, R. E. Solving linear programming problems in integers. *Combinatorial analysis 10*, 211-215 (1960), 25.

[18] Gurobi Optimization LLC. Gurobi optimizer reference manual, 2024.
`https://www.gurobi.com`.

[19] Hall, P. On Representatives of Subsets. *Journal of the London Mathematical Society s1-10*, 1 (1935), 26–30.

[20] Katagiri, H., Qingqiang, G., Bin, W., Muranaka, T., Hamori, H., and Kato, K. Path Optimization for Electrical PCB Inspections with Alignment Operations using Multiple Cameras. *Procedia Computer Science 60* (Jan. 2015), 1051–1060.

[21] Land, A. H., and Doig, A. G. An Automatic Method of Solving Discrete Programming Problems. *Econometrica 28*, 3 (1960), 497–520.

[22] Matai, R., Singh, S., and Mittal, M. L. Traveling salesman problem: An overview of applications, formulations, and solution approaches. In *Traveling Salesman Problem: Theory and Applications*, D. Davendra, Ed. IntechOpen, Rijeka, 2010, ch. 1.

[23] McKay, B. D., and Piperno, A. Practical graph isomorphism, II. *Journal of Symbolic Computation 60* (Jan. 2014), 94–112.

[24] Mosayebi, M., Sodhi, M., and Wettergren, T. A. The Traveling Salesman Problem with Job-times (*TSPJ*). *Computers & Operations Research 129* (May 2021), 105226.

[25] Padberg, M., and Rinaldi, G. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters 6*, 1 (Mar. 1987), 1–7.

[26] Papadimitriou, C. H., and Vempala, S. On The Approximability Of The Traveling Salesman Problem. *Combinatorica 26*, 1 (Feb. 2006), 101–120.

[27] Sahni, S., and Gonzalez, T. P-Complete Approximation Problems. *Journal of the Association for Computing Machinery 23*, 3 (July 1976), 555–565.

[28] Schrijver, A. On the History of Combinatorial Optimization (Till 1960). In *Handbooks in Operations Research and Management Science*, vol. 12. Elsevier, 2005, pp. 1–68.

[29] Shmoys, D. B., and Williamson, D. P. Analyzing the Held-Karp TSP bound: A monotonicity property with application. *Information Processing Letters 35*, 6 (Sept. 1990), 281–285.

[30] Sosso, A. CTSP. Github repository, Oct. 2024. `https://github.com/barabbs/CTSP`.

[31] Williamson, D. P. Analysis of the Held-Karp Heuristic for the Traveling Salesman Problem. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1990.

[32] Wolsey, L. A. Heuristic analysis, linear programming and branch and bound. *Mathematical Programming Studies 13* (1980), 121–134.

[33] Young, C., Johnson, D. S., Smith, M. D., and Karger, D. R. Near-optimal intraprocedural branch alignment. *ACM SIGPLAN Notices 32*, 5 (May 1997), 183–193.