Lab Research Project Thesis

LIX - Ecole Polytechnique

The Distance Geometry Problem as a universal tool for computation

Mael Kupperschmitt

Under the supervision of Prof. Leo LIBERTI

LIX - École Polytechnique

Contents

Abstract

The equivalence between NP-hard problems, established through polynomial reductions, can be leveraged to solve one problem in terms of another. Typically, these "other problems" are those that humans model naturally, such as SAT or Binary Linear Programming (BLP). In contrast, we propose using the Distance Geometry Problem (DGP) as a universal tool for computation. The DGP involves drawing a graph in a K-dimensional Euclidean space, where vertices represent points and edges represent segments between points having length equal to the edge weight. We explore two approaches for reducing the Binary Linear Programming to the Distance Geometry Problem: one that passes through SAT and 3SAT, and another that provides a direct encoding. These results have been derived in the hope that the DGP, through its geometric relaxations, will provide a promising framework for solving BLP instances, highlighting the potential of geometric modeling as a robust and versatile optimization tool.

1 Introduction and Motivations

Binary Linear Programming (BLP) is a well-known combinatorial optimization problem where the goal is to find binary values $x_1, x_2, \ldots, x_n \in \{0, 1\}$ that optimize a cost function subject to a set of linear inequalities. The original BLP problem typically involves a cost vector c and is defined as:

$$\text{Minimize } c^T x \quad \text{subject to} \quad Ax \leq b, \quad x \in \{0,1\}^n,$$

where: - $A \in \mathbb{Z}^{m \times n}$ is a matrix of coefficients representing the linear constraints, - $b \in \mathbb{Z}^m$ is a vector representing the upper bounds for the constraints, - $c \in \mathbb{Z}^n$ is the cost vector, - x is a vector of binary decision variables.

In this work, however, we focus on the feasibility variant of BLP, which asks whether there exists a binary vector x satisfying the constraints. Formally, this feasibility problem is defined as:

$$\text{Find } x \in \{0,1\}^n \quad \text{subject to} \quad Ax \leq b.$$

BLP is a central problem in combinatorial optimization with applications in areas such as network design, scheduling, and resource allocation [2] [4]. However, solving large-scale BLP problems is computationally challenging because BLP is NP-hard by a reduction from SAT [1]. Thus, finding efficient solution methods for BLP instances is of significant practical interest [3].

The Distance Geometry Problem (DGP) is another problem in NP that is defined as follows: given a simple edge-weighted graph $G = (V, E, d)$ and a positive integer K, the goal is to determine if there exists a realization $y : V \to \mathbb{R}^K$ such that the Euclidean distance between pairs of points u and v in G corresponds to the edge weights d_{uv}:

$$\forall \{u, v\} \in E \quad \|y_u - y_v\|_2 = d_{uv}$$

Often, the dimension K is fixed and we then talk about DGP_K. In this report, we explore possible reductions from the BLP to the DGP_1, showing how the latter could be used as a universal tool for computation. Specifically, we aim to "model" BLP instances into DGP_1 instances, leveraging the simple geometric relaxations offered by the DGP_1 to obtain nontrivial lower bounds for the error in the initial BLP instances.

2 Background

2.1 Reduction

A reduction is an algorithm by which every instance of one problem can be transformed into an instance of another problem by which the transformed instance is YES iff the original instance is YES. Moreover by means of a reduction from a problem A to a problem B, a solution to the reduced instance of B can be used to solve the original instance of A. It is desirable that this reduction is a polynomial time reduction meaning that it has polynomial time complexity with respect to the size of the instance of the problem to be reduced.

We will discuss two main reduction approaches: (i) A chain of reduction through SAT: in this approach, BLP is first reduced to SAT, and then SAT is transformed into a DGP by first passing through 3SAT [5]. This method, however, leads to DGP instance which are significantly larger than the initial BLP instances and it quite computationally heavy. (ii) A direct reduction from BLP to DGP: this approach involves directly modeling a BLP instance by means of a DGP instance, bypassing SAT entirely and reducing the size blow up of the DGP instance. While this method is less computationally heavy it requires some additional processing to be able to correctly map the feasibility of the instances, failing to make it a perfect reduction.

2.2 Instances and Certificates

An instance refers to a specific problem setup. An instance of BLP consists of a matrix A and vector b, while an instance of DGP is defined by a pair (G, K), where $G = (V, E, d)$ is a simple edge-weighted graph and K is the dimension of the Euclidean space we need to embed G into.

A certificate is a solution that certifies the feasibility of an instance of a problem. For BLP, it is a binary vector $x \in \{0, 1\}^n$ that satisfies the constraints set by the inequalities. For DGP, the certificate is the set of points $y_1, y_2, \ldots, y_n$ that satisfy the distance constraints.

2.3 MinErrDGP and MinErrBLP

We define two optimization variants to handle approximate solutions and infeasible instances of both problems. In particular this allows us to solve infeasible instance with an infeasible solutions instead of a simple NO:

- MinErrBLP: In the Minimum Error Binary Lineary Programming (MinErrBLP) we seek to minimize the error in satisfying the linear constraints $Ax \leq b$ in a BLP. The goal is to find a binary vector $x \in \{0, 1\}^n$ that minimizes the error:

$$\text{Minimize } \|Ax - b\|_2 \quad \text{subject to} \quad x \in \{0, 1\}^n$$

- MinErrDGP: In the Minimum Error Distance Geometry Problem (MinErrDGP), the goal is to minimize the error between the actual Euclidean distances $\|y_u - y_v\|_2$ and the given distances d_{uv}. Formally, this is defined as:

$$\text{Minimize} \quad \sum_{\{u,v\} \in E} (\|y_u - y_v\|_2 - d_{uv})^2$$

Both variants aim to find the best possible approximate solutions when exact solutions are infeasible.

2.4 Relaxations

A relaxation of a problem is a modification that transforms a problem into a simpler one by loosening some constraints, making it easier to solve. In the case of a BLP instance we can relax the binary vector x so that $x \in [0,1]^n$ instead of $\{0,1\}^n$. The DGP provides a more intrinsic relaxation as we can increment the dimension of the Euclidean space we need to embed our graph in; transforming an instance of DGP_K into an instance of DGP_{K+1}. Note that in the case where the relaxed instance of a problem remains infeasible, the error of this instance becomes a lower bound for the initial instance with all the constraints left intact.

2.5 Error Mapping

Error mapping is important when working with infeasible instances. In the case of MinErrDGP and MinErrBLP error mapping involves translating the errors from a DGP instance back to the BLP instance from which we applied a reduction; allowing us to compute MinErrBLP as a function of MinErrDGP. This is especially useful when we couple this with relaxations, because a lower bound for the MinErrDGP, obtained via geometric relaxation, can serve as a corresponding lower bound for the MinErrBLP of the original instance. This will become even more useful when working with very large and infeasible BLP instances.

3 Passing through SAT

In our first attempt of a reduction to DGP_1 we decided to first reduce to SAT which would allow us to use Saxe's3SATto DGP_1 reduction [5]. The outline of this circuit was already laid out for us however it lacked a serious proof for its linear time complexity, and we wanted to have an implementation that we could use as it would allow us map the errors from the DGP_1 back to the initial BLP. We will first discuss my implementation of Warner's linear time reduction from BLP to SAT [6]; prove Saxe's graph gadgets which reduces 3SAT to DGP_1 [5] and then outline the capacity of the DGP_1 to encode logical operators too.

3.1 Implementation of BLP to SAT

3.1.1 Reduction overview

Given a BLP instance where each constraint is of the form

$$\sum_{i=1}^{n} a_i x_i \text{ rel } b, \qquad \text{where } x \in \{0,1\}^n \wedge \text{rel} \in \{\leq, \geq, =\},$$

our goal is to produce a conjuctive normal form formula $\phi = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n_i} x_{ij} \right)$ such that ϕ is satisfiable if and only if the BLP instance is feasible. It may be noticed that contrary to the initial definition for the BLP we previously introduced, here we allow our sum to be bounded from above or equal to the right hand side. We will quickly see that the there is no difference and that any constraint delimited by $\leq$ or $=$ can be transformed into constraints uniquely delimited by $\geq$.
The key steps to this reduction are:

1. **Normalization:** Transform each constraint so that the relation is $\leq$ and all coefficients are positive. If a coefficient a_j is negative, we flip the literal and the sign of the coefficient and add $|a_j|$ to the bound.

2. **Binary Decomposition:** Represent each positive coefficient a_i in binary. For each bit position k where the kth bit is 1, create an auxiliary variable that is constrained to be equivalent to the original literal.

3. **Binary Summation:** For each bit level, sum the contributions using pairwise XOR (for the sum bit) and AND (for the carry bit). When identical bits cancel, if all contributions cancel then explicitly set the bit to zero.

4. **Inequality Enforcement:** Express the bound b in binary and add clauses that enforce that the computed binary sum does not exceed b.

3.1.2 Pseudo-code

Algorithm 1 NormalizeConstraint

```
procedure NormalizeConstraint(A, b, rel)
    if rel = " = " then
        call NormalizeConstraint(A, b, ≤) and call NormalizeConstraint(A, b, ≥)
    end if
    if rel = " ≥ " then
        for each term (a_i, x_i) do:
            a_i ← −a_i
        end for
        b ← −b
        rel ← " ≤ "
    end if
    for each term (a_i, x_i) in the constraint do
        if a_i < 0 then
            a_i ← |a_i|
            Replace x_i by ¬x_i
            b ← b + |a_i|
        end if
    end for
    return (A, b, rel)
end procedure
```

For constraint given by an equality we transform it into two clauses, one given by a $\leq$ and the other by $\geq$. We can do this since:

$$\sum_{i=1}^{i=n} a_i x_i = b \iff \begin{cases} \sum_{i=1}^{i=n} a_i x_i \leq b \\ \sum_{i=1}^{i=n} a_i x_i \geq b \end{cases}$$

Moreover to make all coefficients positive, we use a neat trick which was not made explicit in Warner's paper [6]. Given that $x_i \in \{0, 1\}$:

$$\forall a_i \in \mathbb{Z}, -|a_i| x_i = (1 - x_i)|a_i| - |a_i| = \neg x_i |a_i| - |a_i|$$

Here below is the pseudo-code for the remaining procedures.

Algorithm 2 BinarySummation

```
1: procedure BinarySummation(S, M, n, encoder)
2:     Initialize carry_bits ← empty dictionary (default value: empty list)
3:     Initialize sum_bits ← empty dictionary
4:     for k = 0 to M + ⌈log_2 n⌉ + 1 do
5:         current_bits ← S[k] ∪ carry_bits[k]                ▷ Cancel duplicate occurrences:
6:         for each distinct bit b in current_bits do
7:             Let c ← count of b in current_bits
8:             Add ⌊c/2⌋ copies of b to carry_bits[k + 1]
9:             if c is odd then
10:                Append b to current_bits′
11:            end if
12:        end for
13:        current_bits ← current_bits′
14:        while |current_bits| > 1 do
15:            (a, b) ← (pop(current_bits), pop(current_bits))
16:            (sum, carry) ← (encoder.new_var(), encoder.new_var())
17:            Add clauses to enforce: sum = a ⊕ b and carry = a ∧ b
18:            Append sum to current_bits
19:            Append carry to carry_bits[k + 1]
20:        end while
21:        if current_bits is empty then
22:            zero ← encoder.new_var()
23:            sum_bits[k] ← zero
24:        else
25:            sum_bits[k] ← current_bits[0]
26:        end if
27:    end for
28:    return sum_bits
29: end procedure
```

Algorithm 3 Enforce the Inequality

```
1: procedure Enforce the Inequality({sum_bit[k]}, b)
2:     Let bound_bits ← binary representation of b
3:     higher_match ← ∅
4:     for k = length(bound_bits) − 1 downto 0 do
5:         if bound_bits[k] = 0 then
6:             if higher_match ≠ ∅ then
7:                 Create a match variable m_k ensuring higher bits match
8:                 Add clause (¬m_k ∨ ¬sum_bit[k])
9:             else
10:                Add clause (¬sum_bit[k])
11:            end if
12:        end if
13:        Create variable match_k such that match_k ≡ (sum_bit[k] = bound_bits[k])
14:        Add match_k to higher_match
15:    end for
16:    for k > length(bound_bits) − 1 do
17:        Add clause (¬sum_bit[k])
18:    end for
19: end procedure
```

3.1.3 Proof of linear time complexity

We now argue that the overall reduction runs in linear time with respect to the input size (i.e., the total number of terms in the BLP instance or in the coefficient matrix $A \in Mat_{m,n}(\mathbb{Z})$), under the assumption that the coefficients are bounded.

Normalization Step:
Each constraint is processed by iterating over its terms once. Thus, if there are nm terms in total, normalization takes $O(nm)$ time. If the relation is $=$ we iterate over each term twice, still make the time complexity $O(nm)$.

Binary Decomposition and Summation:
For each term, we examine up to $M = \lceil \log_2(\max_i a_i) \rceil$ bit positions. With coefficients bounded, M is a constant.

- Creating the auxiliary variables and adding the correspondWing equivalence clauses requires constant time per bit.
- The pairwise cancellation and summation across each bit level processes each auxiliary variable at most once, and even if we perform a pairing step that takes time proportional to the number of bits in that level, the overall time is $O(nm)$ because M is constant.

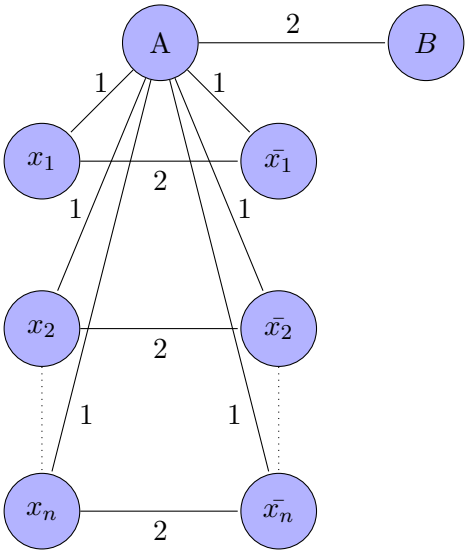
Enforcing the Inequality:
This step iterates over the bits of b. Again, since b is at most the sum of the coefficients, and the coefficients are bounded, the number of bits is constant. Thus, this step is also $O(nm)$.

Conclusion:
Since each stage of the reduction runs in time linear in the number of terms (or, more generally, linear in the size of the BLP instance), the entire transformation is linear time. This forms the basis for an efficient polynomial-time reduction from BLP to SAT.

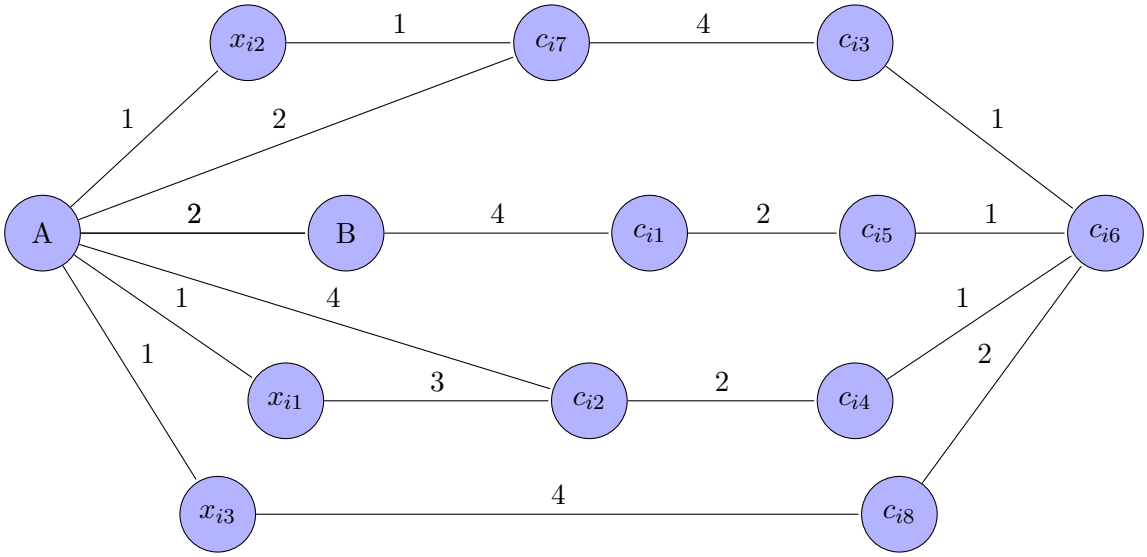
3.2 Proving Saxe's reduction from 3sat to DGP_1

The reduction from SAT to 3SAT is well known and rigorously proven [1] so we knew that we could chain the reduction and obtain 3SAT instances from our BLP instances. In his paper in 1979 [5], Saxe showed a graph gadget (shown here below, Figure. 1a) that would encode 3SAT clauses into a DGP instance, however he did not provide any formal proof for his statement stating that "Careful study of the graph [..] will reveal that it is impossible to embed it in the line in such a way that A is sent to 0, B is sent to 2, and all three x_{ij} are sent to -1 (FALSE), but if one or more of the x_{ij} are sent to 1 (TRUE) then an embedding is possible (in fact only one such embedding is possible)". This was in fact nontrivial but here below, I offer a proof for the following theorem.

Theorem 1. *The construction proposed below yields a valid polynomial reduction from an instance τ of* 3SAT *to a corresponding instance γ of DGP_1. Moreover, we have $x_{ij} = 1 \wedge x_{\bar{ij}} = -1$ in the realization solving γ if and only if the* 3SAT *variable x_j is TRUE, and $x_{ij} = -1 \wedge x_{\bar{ij}} = 1$ iff x_j is FALSE.*



(a) The gadget for literals used in the $3SAT \rightarrow DGP_1$ reduction.



(b) The gadget for clause i used in the $3SAT \rightarrow DGP_1$ reduction.

Figure 1: Graph gadgets used in the $3SAT \rightarrow DGP_1$ reduction.

Proof. The theorem asserts that the graph gadgets in Figures 1a and 1b correctly encode the 3SAT clauses into a DGP_1 instance.

- The first figure (Figure. 1a) encodes the literals of the 3SAT formula. For each literal two vertices are created, one encoding the literal and the other its negation. As the A and B vertices are respectively fixed at the origin and at 2 and we see that each vertex encoding the literals can either be at -1 or at 1 because there is an edge weight of length 2 between the vertices encoding a literal and its negation. If the vertex $\bar{x_i}$ is at 1 this means the literal x_i is false in the clause of the original 3SAT instance and if the vertex x_i is at 1 this means that the literal x_i is true in the clause of the original 3SAT instance.
- The second figure (Figure. 1b) encodes the clauses of the 3SAT formula, where the literals from the first figure are connected appropriately based on the structure of the clauses. We consider a 3SAT

clause of the form $(x_{i1} \vee x_{i2} \vee x_{i3})$. Let us show that this reduction correctly maps infeasible 3SAT instances to an infeasible DGP_1 instance. We thus prove that we cannot embed this graph gadget in one dimension when all vertices x_{i1}, x_{i2}, x_{i3} are positioned at -1. For the sake of contradiction let us assume that we can embed this graph when x_{i1}, x_{i2}, x_{i3} are all positioned at -1. Consider the four paths $P_1 = (A, x_{i2}, c_{i7}, c_{i3}, c_{i6})$, $P_2 = (A, B, c_{i1}, c_{i5}, c_{i6})$, $P_3 = (A, x_{i1}, c_{i2}, c_{i4}, c_{i6})$ and $P_4 = (A, x_{i3}, c_{i8}, c_{i6})$ which all end at the vertex c_{i6}. As the graph can be embedded, regardless of the path we take, the c_{i6} vertex must be at the same location, this leads us with several options as to where we can place the c_{i6} vertex. Notice the edges (A, c_{i7}) and (A, c_{i2}) which due to their weight (2 and 4 respectively) and the fact that $x_{i1} = x_{i2} = -1$, forces us to position c_{i7} and c_{i2} at -2 and -4 respectively in order to embed the graph in 1 dimension. Using this, the edge weight of the vertices in these four paths and the fact that the vertices A and B are at position 0 and 2 respectively we compute what position of c_{i6} each path allows. P_1 allows us to position c_{i6} at $-1-1\pm4\pm1 = \{-7, -5, 1, 3\}$. P_2 allows us to position c_{i6} at $0+2\pm4\pm2\pm1 = \{\pm5, \pm3, \pm1, 7, 9\}$. P_3 allows $-1 - 3 \pm 2 \pm 1 = \{-7, -5, -3, -1\}$ and P_4 allows $-1 \pm 4 \pm 2 = \{-7, -3, 1, 5\}$. The position of c_{i6} must satisfy the constraints set by each of these paths however the intersection of these four sets is the empty set. Thus meaning that there is no plausible candidate for the position of c_{i6} that would make the graph embeddable in 1 dimension. We have thus a contradiction and the graph cannot be embedded in 1 dimension. Hence this reduction correctly maps infeasible 3SAT instances to infeasible DGP instances.

- Now let us handle feasible instances. We consider all possible YES clauses of a 3SAT $(x_{i1} \vee x_{i2} \vee x_{i3})$ under the form of a tuple $(x_{i1}, x_{i2}, x_{i3}) \neq (-1, -1, -1)$ and offer a correct embedding of all the vertices of the graph gadget under the form of another tuple $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$.
  Here below find all possible cases and the position of all vertices which make the graph embeddable in 1 dimension.

  Tuple: $(c_7, c_3, c_1, c_5, c_2, c_4, c_8, c_6)$

  - $(x_{i1}, x_{i2}, x_{i3}) = (1, 1, 1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (6,4,6,6,5,7,2,5)
  - $(x_{i1}, x_{i2}, x_{i3}) = (1, 1, -1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (6,4,6,6,4,5,2,3)
  - $(x_{i1}, x_{i2}, x_{i3}) = (1, -1, 1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (6,4,2,2,4,3,-2,5)
  - $(x_{i1}, x_{i2}, x_{i3}) = (-1, 1, 1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (-2,-4,-2,-2,0,-1,2,-3)
  - $(x_{i1}, x_{i2}, x_{i3}) = (1, -1, -1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (-2,4,2,2,0,1,-2,3)
  - $(x_{i1}, x_{i2}, x_{i3}) = (-1, 1, -1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (-2,-4,-4,-2,-4,-3,2,-5)
  - $(x_{i1}, x_{i2}, x_{i3}) = (-1, -1, 1)$ then $(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}, c_{i8})$ = (-2,-4,-6,-6,-4,-5,-2,-3)

Thus, the graph gadgets correctly encode the 3SAT problem into a DGP_1 instance. □

3.3 Encoding Logical Operators via DGP Gadgets

Inspired by Saxe's construction for encoding 3SAT clauses, I explored how similar gadgets could be used to geometrically model basic logical operations using the DGP framework. The goal is to design a graph such that it is embeddable in $\mathbb{R}^K$ if and only if a logical condition of the form $x_1 \lambda x_2 = 1$ holds, where $\lambda \in \{\vee, \wedge, \oplus\}$ represents a logical OR, AND, or XOR operation. Each gadget shares the same structure: a fixed anchor vertex A at position 0, input vertices x_1 and x_2, and a node c_6 which is a common node in 3 different cycles. Again, just like in Saxe's reduction, the literals are consider to be TRUE when the associated vertex is found at +1 and FALSE when the vertex is at -1. The graph is constructed so that the cycles to c_6 become geometrically infeasible unless the inputs satisfy the

desired logic condition. Figures 2, 3, and 4 illustrate the gadgets encoding the AND, OR, and XOR operations, respectively.

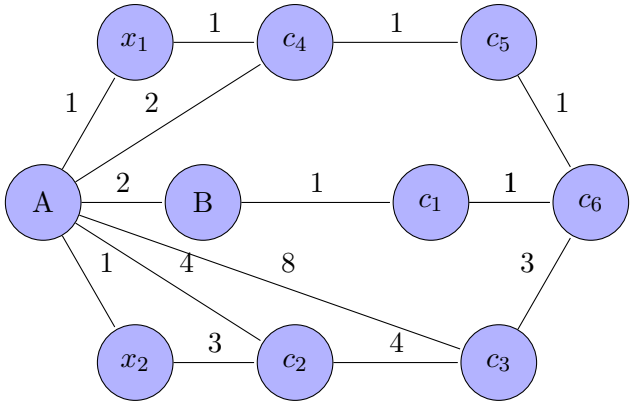


Figure 2: Encoding the AND operator

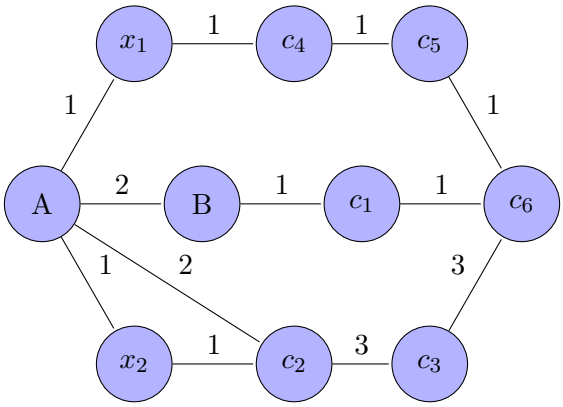


Figure 3: Encoding the OR operator

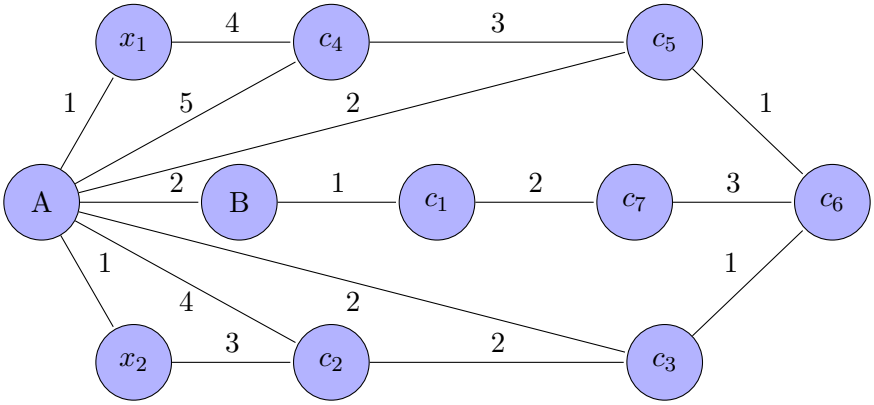


Figure 4: Encoding the XOR operator

4 An attempt at a direct transformation

While the reduction gadgets via SAT provide a functional transformation from BLP to DGP, it comes with significant drawbacks: the constructed instance is large, often combinatorially complex, and it severely distorts the structure of the original BLP. As a consequence, the error mapping from MinErrDGP back to MinErrBLP for infeasible cases becomes very difficult, especially for large cases. Motivated by this, I attempted a more direct transformation from BLP to DGP, aiming to preserve more of the original structure and potentially simplify the mapping of error metrics. This approach builds custom gadgets for constraints and variables, avoiding the intermediate logic encoding entirely. While this is not a formal polynomial-time reduction, it represents a theoretical probing of the computational power of the DGP. In the following section, I describe the full construction logic of the graph building process.

Here we attempt to encode a normalized (all coefficients are positive) BLP constraint of the form $\sum_{i=j}^{n} x_j a_j \leq b$ where $a_j \in \mathbb{N}\ \forall 1 \leq j \leq n$.
Here below are the different important graph gadgets we are using with each other. Each gadget has a specific role that it handles in the reduction. Below these I provided an example graph reduction for a simple clause containing 2 literals.

4.1 Encoding the literals and the bound

Here we start by encoding the literals of our BLP clause, a sequence of partial sums and the bound. For simplicity, the vertices A and P_0 are positioned at 0 and 1 respectively. With regards to the encoding of the literals, we use similar system to the one Saxe used in his encoding of a 3SAT clause. However here, while the vertex corresponding to the literal can be positioned at -1 or at 1 we do not create a vertex corresponding to the negation of the literal. Just like in the reduction from 3SAT, a literal is considered 'active' when it is found at position +1. We would thus like to create a mechanism that forces the partial sum to be extended by a_j when the literal x_j is embedded at 1 and that the partial sum is not extended when the literal x_j is embedded at -1.
For this reduction to work we need to specify a distinction between directed and undirected edges in our graph. In regular DGP_1 instances each edges are undirected and can be folded in two different ways (either to the left or to the right), however here I have decided to implement directed edges which can only be folded in one direction. In all of the following graph diagrams, directed edges can be distinguished with arrows and the direction of the folding is made explicit with the direction of the arrow (i.e in figure5 the vertex M1 must be placed at position $+(\frac{a_1}{2}+1)$. Thanks to these new features I have decided to use, notice that each vertex P_j can be extended either by $+a_j$ or by 0 w.r.t to the previous vertex P_{j-1}. The relation between each P_j vertex and the vertices corresponding to x_j is further detailed in Figure 7 and Figure 8.
Finally we encode the bound by adding a directed edge of weight b from our initial vertex P_0 to a target vertex T. This makes it impossible for our partial sum to exceed the value of b while keeping the DGP instance feasible. While a normalized BLP clause asks for the sum of the literals and coefficients to be smaller or equal to specific bound, we cannot explicitly encode this with DGP constraints as each edge weight must be an exact number. We solve this by adding a buffer $\delta \geq 0$ between the final node of the partial sum P_n and the target vertex T. This buffer will be detailed with Figure 6.

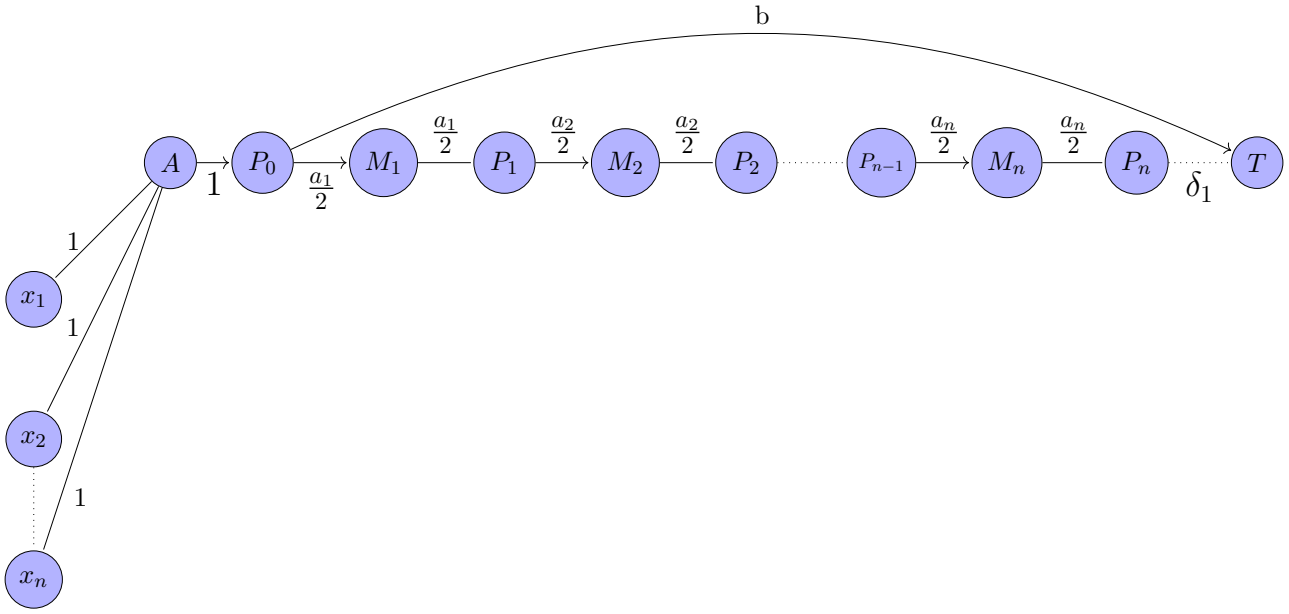


Figure 5: Encoding the literals, the partial sum and the bound

4.2 Encoding the buffer

The following graph (Figure 6) illustrates the buffer mechanism used to encode the remaining part of the sum of the literals and coefficients in a BLP clause. The buffer is designed to accommodate any leftover value in the constraint sum, up to a given bound b. To achieve this, we use the parameter $l = \lfloor \log_2 b \rfloor$, which determines the number of pairs of vertices (R_j, S_j) created for $0 \leq i \leq l$. These pairs are crucial in encoding numbers within the range of the buffer.

Each pair (R_j, S_j) works similarly to the pairs (M_j, P_j) shown in Figure 5. Just like in Figure 5, each pair (R_j, S_j) is designed to increment the sum by either 2^{j-1} or 0. However, unlike the pairs in Figure5, whether these pairs increment the sum or not is not determined by the position of certain corresponding vertices. These pairs are completely independent from the rest of the graph and the configuration ensures that any integer between 0 and b can be encoded by systematically extending the sum with contributions from each pair (R_j, S_j). The buffer's functionality is guaranteed as long as the sum of the previous edges before the buffer does not exceed the bound b (i.e. $P_n \leq T$). Once this condition is met, the buffer can complete the remaining part of the sum and ensure that the total sum is precisely equal to b. Importantly, as just explained, these buffer vertices R_j and S_j are independent of the rest of the graph, meaning they can be placed without disturbing the rest of the structure as long as the P_n vertex remains positioned at a value less than or equal to b. This guarantees that the vertices in the buffer can be embedded properly without causing conflicts or increasing the potential error.

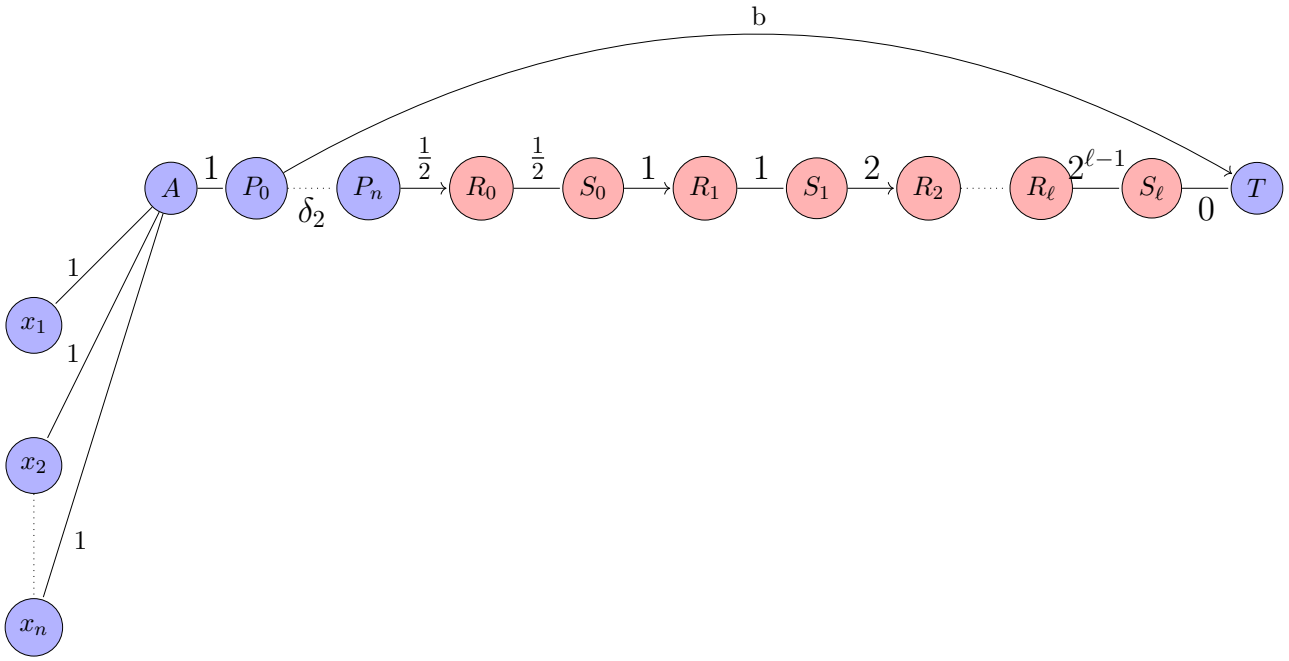


Figure 6: Encoding the buffer to the bound

4.3 Encoding the link between the literals and the sum

For the reduction to work, we need a mechanism that sets the vertex x_j to 1, if and only if the coefficient a_j contributes to the sum meaning $P_j = P_{j-1}$ (i.e., $x_j = 1$ in the original BLP clause). Conversely, the mechanism should set x_j to -1 if and only if the coefficient a_j does not contribute to the sum meaning $P_j = a_j + P_{j-1}$ (i.e., $x_j = 0$ in the original BLP constraint).

Initially, I thought of implementing this mechanism directly. However it seemed almost impossible, forcing me to take on a different approach. Instead of trying to make my gadget embeddable, meaning that it would not increase the total MinErrDGP of my graph, I decided to formulate a mechanism that would always create an error. However the error would be minimized when:

$$x_j = B_j + 1 \wedge P_j = a_j + P_{j-1} \quad \text{or} \quad x_j = B_j - 1 \wedge P_j = 0 + P_{j-1}$$

and maximizes the error when:

$$x_j = B_j + 1 \wedge P_j = 0 + P_{j-1} \quad \text{or} \quad x_j = B_j - 1 \wedge P_j = a_j + P_{j-1}$$

The idea is that the error is minimized when the values of x_j and P_j are consistent with their desired behaviour, and the error is maximized when they are inconsistent. It is also important for our reduction that the $DGPerr(x_j = 1 \wedge P_j = a_j + P_{j-1}) = DGPerr(x_j = -1 \wedge P_j = 0 + P_{j-1})$ that way, when minimizing the error, our system will have no preference between the two 'correct' ways our gadget can embed itself in.

Thus, by choosing the DGP constraint that minimizes the error, we can find the configuration that correctly associates the literals with the values they contribute to the sum, mimicking the correct behavior of BLP constraints. So that there is no ambiguity, let us define the error between two vertices A and B, connected by an edge of weight w, as the absolute difference between the distance $dist(A, B)$ in the embedded graph and the edge weight w:

$$\text{Error}(A, B) = |dist(A, B) - w|$$

Based on this definition, the MinErrDGP problem previously introduced can be written as:

$$\min_{x \in \mathbb{R}^{NK}, s \geq 0} \left\{ \sum_{\{u,v\} \in E} \left(s_{uv}^+ + s_{uv}^-\right) \mid \forall \{u, v\} \in E,\ \|x_u - x_v\|_2^2 - d_{uv}^2 = s_{uv}^+ - s_{uv}^- \right\}$$

This formulation minimizes the total error, ensuring that the graph is embedded in such a way that the edge lengths reflect the correct contributions of the literals to the sum, in line with the constraints of the original BLP problem:
$\min_{x \in \mathbb{R}^{NK}, s \geq 0} \left\{ \sum_{\{u,v\} \in E} \left(s_{uv}^+ + s_{uv}^-\right) \middle| \forall \{u, v\} \in E,\ \|x_u - x_v\|_2^2 - d_{uv}^2 = s_{uv}^+ - s_{uv}^- \right\}$

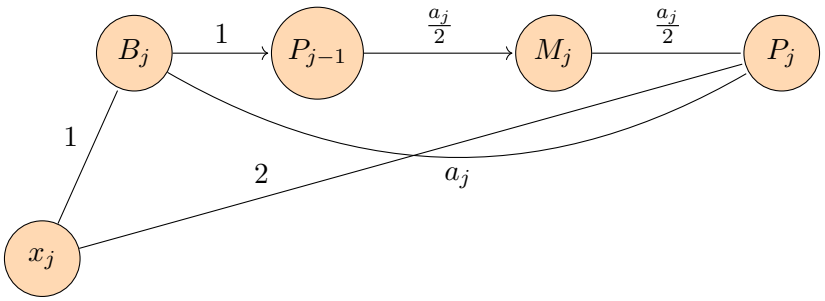


Figure 7: Encoding the relationship between the literals and the sum

Proposition 1: The above gadget ensures that the following conditions holds for all $a_j \in \mathbb{N}^*$:

1.
$$\text{DGPerror}(x_j = B_j + 1, P_j = a_j + P_{j-1}) < \text{DGPerror}(x_j = B_j + 1, P_j = 0 + P_{j-1})$$

2.
$$\text{DGPerror}(x_j = B_j - 1, P_j = 0 + P_{j-1}) < \text{DGPerror}(x_j = B_j - 1, P_j = a_j + P_{j-1})$$

3.
$$\text{DGPerror}(x_j = B_j - 1, P_j = 0 + P_{j-1}) = \text{DGPerror}(x_j = B_j + 1, P_j = a_j + P_{j-1})$$

Proof. Let us prove that the gadget satisifies each condition using the formula defined above for the error between two vertices:

Condition1:

$\text{DGPerror}(x_j = B_j{+}1, P_j = a_j{+}P_{j-1}) = |(a_j{+}1{-}1){-}2|{+}|(a_j{+}1){-}a_j| = |a_j{-}2|{+}1 = a_j{-}1$ for all $a_j \in \mathbb{N} \setminus \{0\}$

$$\text{DGPerror}(x_j = B_j + 1, P_j = 0 + P_{j-1}) = |0 - 2| + |1 - a_j| = |a_j - 1| + 2 = a_j + 1$$

Since $a_j - 1 < a_j + 1$ we clearly meet the required condition

Condition 2:

$$\text{DGPerror}(x_j = B_j - 1, P_j = a_j + P_{j-1}) = |(a_j + 1 + 1) - 2| + |(a_j + 1) - a_j| = a_j + 1 = a_j + 1$$

$\text{DGPerror}(x_j = B_j{-}1, P_j = 0{+}P_{j-1}) = |2{-}2|{+}|1{-}a_j| = 0{+}|a_j{-}1| = a_j{-}1$ since $a_j \in \mathbb{N}$ and $a_j \geq 1$

Again, since $a_j - 1 < a_j + 1$ we clearly meet the required condition

Condition 3:
From the previous computations we have found that:

$$\text{DGPerror}(x_j = B_j + 1, P_j = a_j + P_{j-1}) = a_j - 1 = \text{DGPerror}(x_j = B_j - 1, P_j = 0 + P_{j-1})$$

□

We have thus found a mechanism that, when minimizing the error, will correctly extend the partial sum by a_j when the x_j vertex is at +1 w.r.t B_j and will not extend the partial sum when the x_j vertex is at -1 w.r.t B_j. While the gagdet in Figure7 shows the literals connected to an B_j vertex next to the P_j vertex, all of our vertices encoding the literals are connected to the A vertex fixed at 0 and situated next to the P_0 vertex. We cannot change this initial configuration as we need to be able to reduce multiple BLP constraints which contain the same literals. To solve this problem, we create an auxiliary vertex x_j^i for each literal x_j in the i-th clause who will (instead of x_j) decide whether the sum is extended by a_j or not due to the gadget in Figure7. Note that if $a_j = 0$ we will not create this link as it is not necessary and actually distorts the functioning of the gadget in Figure 7. In order to 'connect' the x_j vertex to the corresponding x_j^i vertex we create a path $(N_k, P_k)_{1 \leq k \leq j-1}$ that can mimic the behaviour of the path of the partial sums. This new path from x_j to x_j^i will embed itself in exactly the same way the path from P_0 to P_{j-1} does. To embed this path, the x_j^i vertex will be embedded in the same position w.r.t to B_j as x_j is w.r.t to A. Hence without being directly connected to the B_j vertex, the position of x_j will determine whether the P_j vertex extends, extending the sum by a_j, or folds back to P_{j-1}, extending the sum by 0.

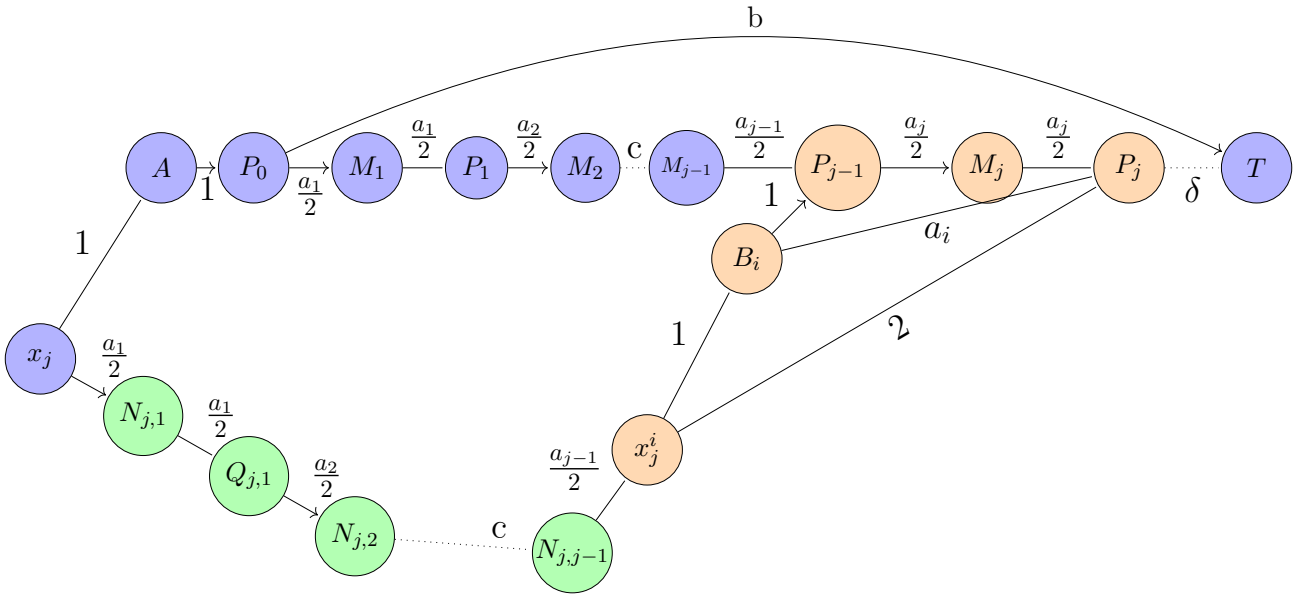


Figure 8: Encoding the auxiliary literal extension mechanism

4.4 How this reduction works and how to use it

Let us now discuss precisely how this graph gadget manages to reduce a normalized BLP instance of the form:

$$\begin{array}{rcl} \sum_{j=1}^{n} a_{j1}x_j & \leq & b_1 \\ \sum_{j=1}^{n} a_{j2}x_j & \leq & b_2 \\ \sum_{j=1}^{n} a_{j3}x_j & \leq & b_3 \\ \vdots & & \vdots \\ \sum_{j=1}^{n} a_{jm}x_j & \leq & b_m \end{array}$$

where:

$$\text{for all } 1 \leq j \leq n \quad x_j \in \{0,1\} \text{ and for all } 1 \leq j \leq n \text{ and } 1 \leq i \leq m \quad a_{ji} \in \mathbf{N}.$$

For clarity, we will say that each BLP constraint is mapped through this graph gadget to its respective DGP constraint which are all superposed with each other in 1 dimension. For each $1 \leq i \leq m$ we use the above graph gadget to reduce the i-th BLP constraint in the instance into a DGP constraint as shown below. For simplicity I have just shown how an arbitrary x_j literal (assuming that $a_{ji} \geq 1$ behaves w.r.t to the rest of the graph:

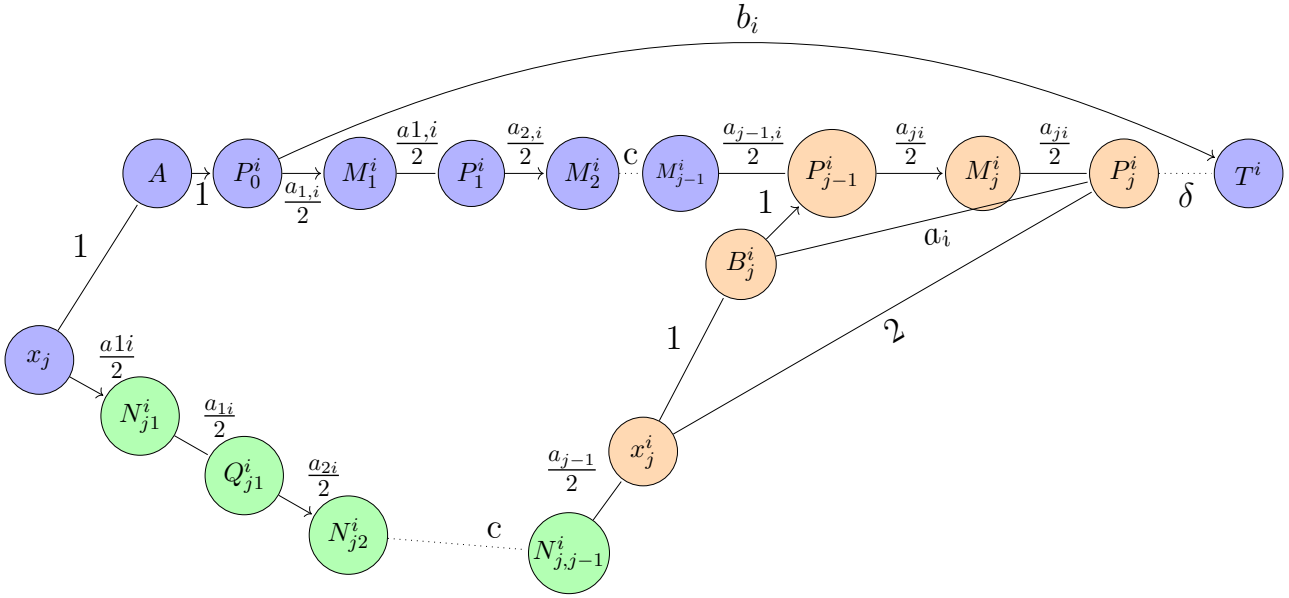


Figure 9: Encoding the i-th constraint of a DGP instance

This reduction is not direct and does not directly map feasible BLP instances to feasible DGP instances or infeasible BLP instances to infeasible DGP instances. In fact, due to the error caused by the mechanism discussed in Figure7, this reduction will always map a BLP instance to an infeasible DGP instance. We still have some computation to do before this graph gadget becomes useful. Note that the only time errors may arise in the reduced DGP instance is in the mechanism discussed in Figure7, in the literal extension paths $(N_{j,k}^i, Q_{j,k}^i)_{1 \leq k \leq j-1}$ for each literal x_j and if the distance between the P_0^i and P_n^i vertex exceeds b_i (meaning the buffer would have to fold backwards). All the other vertices can be trivially embedded and will not cause the error of the DGP instance to increase. Taking all possible embeddings of the infeasible DGP instance, we select the embedding that minimizes the error:

$$\min_{x \in \mathbb{R}^{NK}, s \geq 0} \left\{ \sum_{\{u,v\} \in E} \left(s_{uv}^+ + s_{uv}^-\right) \mid \forall \{u,v\} \in E,\ \|x_u - x_v\|_2^2 - d_{uv}^2 = s_{uv}^+ - s_{uv}^- \right\}$$

Thanks to the mechanism discussed in Figure7, we know this embedding will encode the correct relationship between the position of each x_j vertex and the extension of the partial sum in the i-th DGP constraint by a_{ji} or 0. We know this as we just showed that this mechanism in Figure7 minimizes the error when the relationship between the position of the x_j and P_j^i vertices mimicks the one found in the initial BLP instance. This embedding will also fold the literal extension paths $(N_{j,k}^i, Q_{j,k}^i)_{1\leq k\leq j-1}$ exactly like the partial sum path $(M_k^i, P_k^i)_{1\leq k\leq j-1}$ for each literal x_j. When finding the embedding that minimizes the error, we also get the embedding that least overextends the partial sum of $(P_j)_{1\leq i\leq n}$ w.r.t to the T vertex for each constraint while maintaining the correct behaviour between the x_j and P_j vertices. In fact, we obtain the embedding that minimizes the following set: $\left\{\langle 1, s\rangle | P_n^i \leq T^i + s_i, 1 \leq i \leq n\right\}$, where P_n^i and T^i are the positions of the P_n^i and T^i vertices of the i-th DGP constraint. Notice how similar this is to the MinERR-BLP problem which minimizes the set: $\{\langle 1, s\rangle \mid Ay \leq b + s\}, y \in \{0,1\}^n$ and where $s \in (\mathbb{R}_+)^n$.
In fact it is clear that, when using the graph reduction discussed above we have:

$$\min\{\langle 1, s\rangle \mid Ay \leq b + s\} = \min\left\{\langle 1, s\rangle | P_n^i \leq T^i + s_i, 1 \leq i \leq n\right\} \tag{1}$$

Taking the embedding that minimizes the our DGP error, we can use it to obtain a solution to the MinERR-BLP problem of our initial BLP instance. Denote by E the error of this embedding. Note that for this embedding, $\text{DGPerror}(x_j = 1, P_j^i = a_{ji} + P_{j-1}^i) = a_{ji} - 1 = \text{DGPerror}(x_j = -1, P_j^i = 0 + P_{i-1})$ for all $1 \leq i \leq m$ (as showed using Figure7). We thus have the following formula for the error E:

$$E = (\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ji} - 1)\mathbf{1}_{aji\geq 1}) + \min\left\{\langle 1, s\rangle | P_n^i \leq T^i + s_i, 1 \leq i \leq m\right\} \text{ where } \mathbf{1} \text{ is the indicator function.} \tag{2}$$

Hence using (1)

$$\min\{\langle 1, s\rangle \mid Ay \leq b + s\} = \min\left\{\langle 1, s\rangle | P_n^k \leq T^i + s_i, 1 \leq i \leq m\right\} = E - (\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ji} - 1)\mathbf{1}_{a_{ji}\geq 1}) \tag{3}$$

This reduction not only allows us to determine if the initial BLP instance is feasible or not (as a feasible instance will have a MinErrBLP of 0) but in the case the BLP instance is infeasible, we can easily compute the MinERR-BLP for this instance.

5 Other transformation to DGP_2 variant

The above transformation is successful in the sense that we barely distort the BLP instance and we have managed to quite accurately mimick the behaviour of a BLP in the DGP world. However, the transformation works by taking the embedding which minimizes the DGP error, a process which comes with a lot of computational overhead. Moreover this tranformation is not the perfect theoretical reduction we were originally looking for since we are always mapping to infeasible DGP instances. As already discussed, this comes from the gadget explained in Figure 7 which creates the relationship between the position of the vertex x_j and the extension of the partial sum by a_j or 0. While it may be possible, I have not found a DGP_1 instance that creates this link while being embeddable in 1 dimension. However I have found a slight variant of the graph gadget in Figure 7 that avoids introducing error altogether by lifting some parts of the embedding into the second dimensions.

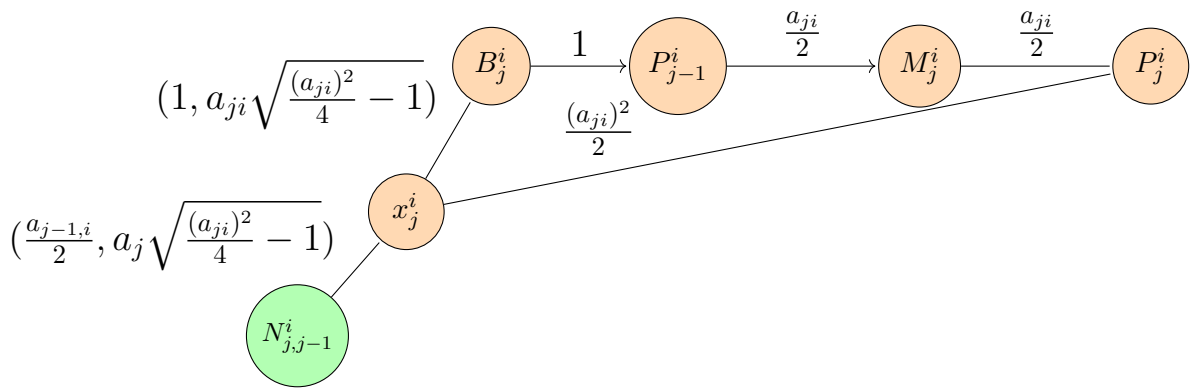


Figure 10: Encoding the relationship between the literals and the sum without an error term but by using another dimension

Here above is another mechanism that encodes the link between the literal x_j and the extension of the partial sum that is embeddable in 2 dimensions. While this only works for $a_j \geq 2$, simply multiplying the initial BLP constraint by 2 before starting the reduction procedure solves this problem. The very reason we are interested in studying DGPs is due to the natural geometric relaxations it allows. With this second gadget, this is exactly what we use. Fixing the partial sum on a 1 dimensional line we distance the x_j^i vertex $a_{ji}\sqrt{(a_{ji}{}^2/4) - 1}$ into the second dimension above (or below it does not change anything, but for simplicity we will fix them above). To make this more understandable, I have introduced a new notation to describe distances in 2 dimensions: the distance between x_j^i and B_j^i is $(1, a_{ji}\sqrt{(a_{ji}^2/4) - 1})$ meaning that their distance in the 1st dimension is 1 but their distance in the 2nd dimension is $a_{ji}\sqrt{(a_{ji}{}^2/4) - 1}$. This notation is not strictly necessary since we have fixed the x_j^i vertex on a line above the rest of the partial sum and I could have simply written the distances as $\sqrt{1 + a_{ji}{}^2(a_{ji}^2/4W - 1)} = \sqrt{(a_{ji}^2 - 2)^2/4} = \frac{a_{ji}^2 - 2}{2}$ and $\sqrt{(a_{j-1,i}^2/4) + a_{ji}^2(a_{ji}^2/4 - 1)}$ respectively. Using Pythagoras' theorem and looking at the paths (B_j^i, x_j^i, P_j^i) and $(B_j^i, P_{j-1}^i, M_j^i, P_j^i)$ is embeddable when there is the correct relationship between the literal and the partial sum::
When x_j^i is to the left of B_j^i and $P_j^i = P_{j-1}^i$ the cycle holds up as:
$\sqrt{a_{ji}{}^2(a_{ji}{}^2/4 - 1) + 1} + 1 = \sqrt{(\frac{a_{ji}{}^2}{2} - 1)^2} + 1 = \frac{a_{ji}{}^2}{2} - 1 + 1 = \frac{a_{ji}{}^2}{2}$

When x_j^i is to the right of B_j^i and $P_j^i = a_{ji} + P_{j-1}$ the cycle holds up as:
$\sqrt{a_{j,i}^2(\frac{a_j^i{}^2}{4} - 1) + a_{ji}^2} = \sqrt{\frac{a_{ji}{}^4}{4}} = \frac{a_{ji}{}^2}{2}$
This transformation encodes the logic of the BLP constraint without forcing an error. While this no longer fits the original DGP_1 model (where all points lie in $\mathbb{R}^1$), it is a valid instance of a relaxed variant of a DGP_2 with partial dimensional fixing: the partial sum is fixed along a line in $\mathbb{R}^2$ and the literal nodes are placed at a fixed offset in the orthogonal dimension. This is not a classical DGP_2 instance, as we control the vertical placement of some points externally. Nonetheless, it provides a compelling proof of concept: the feasibility of a BLP instance can be encoded into a distance constraint system. Since we are fixing the position of some vertices in a second dimension, using this mechanism to control the extension of the partial sum instead of the one described in Figure 7 would be making us reduce to a variant of the DGP_2 and not to the DGP_1.

6 Discussion

Implementing JP Warners linear time reduction [6] from BLP to SAT and proving Saxe's reduction from 3SAT to DGP_1 [5] has solidified the pre-existing blueprint of using the DGP as a tool to solve BLP instances. A precise error mapping through this reduction chain would further solidify this method and make it usable. Nevertheless, due to the considerable distortion of the BLP instance through this reduction chain, finding an error mapping presents itself as a complicated task and the mapping itself should be quite complex. Encoding the logical operators with the DGP_1 is also an interesting idea which I have not fully explored yet but which warrants further attention.
The direct transformation from BLP to DGP_1 detailed in section3 has shown how we can use the DGP to model the BLP in a more direct approach. The simplicity in the error mapping using this transformation leads to believe that it might have an experimental advantage to the reduction chain passing through SAT. While we have not yet found a transformation to the DGP_1 that correctly maps the feasibility of the initial BLP instances, the graph gagdets in Figure 10 is promising and motivates further research. I have implemented the transformation in a python script but it remains to be tested as we would need to adapt and fix the Branch and Prune algorithm developed by another bachelor student in his thesis which would allow us to efficiently solve these DGP_1 instances.

7 Conclusion

In this report, we investigated the use of the Distance Geometry Problem (DGP) as a framework for modeling and solving Binary Linear Programming (BLP) problems. Two main approaches were explored: an indirect reduction passing through SAT and 3SAT, and a more direct transformation from BLP to DGP_1.

The SAT-based method, while rigorous, introduces significant structural overhead, making it difficult ‘to trace back the error in the original BLP instance. This motivated me to attempt a design of a direct modeling strategy that aims to preserve more of the original structure of the problem.

We proposed a new construction using geometric gadgets that encode BLP constraints directly in a DGP instance. Although this approach does not constitute a formal polynomial-time reduction, it offers a more understandable modeling of the initial BLP instance in a DGP framework. The graph gadget in Figure 10 motivates me to believe that a perfect polynomial time reduction from BLP to DGP_1 is possible. Although I am aware of the instance size blow-up implied by these transformations to the DGP_1, it is unlikely that we can solve the DGP_1 instances faster than their BLP counterparts. Nevertheless, the DGP context offers the possibility of deriving easy relaxations by increasing the spacial dimension, the values of which could then be mapped back to the BLP through error mapping techniques that must still be further developed.

These results suggest that the DGP is not only a relevant tool for geometric applications, but also a potential modeling environment for certain classes of combinatorial problems. This work serves as an initial exploration of that idea, and points toward further research both theoretical and experimental into how geometric relaxations could support new optimization techniques.

References

[1] Richard M. Karp. *Reducibility among Combinatorial Problems*. Springer US, Boston, MA, 1972.

[2] Przemysław Kowalik and Magdalena Rzemieniak. Binary linear programming as a tool of cost optimization for a water supply operator. *Sustainability*, 13(6), 2021.

[3] George Nemhauser and Laurence Wolsey. *The Scope of Integer and Combinatorial Optimization*, chapter I.1, pages 1–26. John Wiley Sons, Ltd, 1988.

[4] Samarathunga Samarathunga and Lahiru Wellahetti. A binary linear programming model for a case study in university timetabling problem, 09 2023.

[5] J. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. In *Proceedings of 17th Allerton Conference in Communications, Control and Computing*, 1979.

[6] Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.