# Optimizing the ecological connectivity of landscapes

François Hamonic, Cécile H. Albert, Basile Couëtoux, Yann Vaxès

# Optimizing the ecological connectivity of landscapes

François Hamonic[1,2], Cécile Albert[2], Basile Couëtoux[1], and Yann Vaxès[1]

[1]Aix-Marseille Univ, CNRS, LIS, Marseille, France
[2]Aix-Marseille Univ, CNRS, Univ Avignon, IRD, IMBE, Marseille, France
*francois.hamonic@lis-lab.fr, cecile.albert@imbe.fr, basile.couetoux@lis-lab.fr, yann.vaxes@lis-lab.fr*

**Abstract**

In this article, we consider the problem of optimizing the connectivity of a landscape under a budget constraint, by improving habitat areas and ecological corridors between them. We model this problem as a discrete optimization problem over graphs, in which vertices represent the habitat areas and arcs represent the connections between them. We propose a new flow-based integer linear programming formulation that improves upon the existing models for this problem. By following an approach similar to Catanzaro et al. [7] for the robust shortest path problem, we design an improved preprocessing algorithm that reduces the size of the graphs on which we compute generalized flows. Computational experiments show the benefits of both contributions, by enabling to solve instances of the problem larger than previous models. These experiments also show that several versions of greedy algorithms perform relatively well in practice, while returning arbitrarily bad solutions in the worst case.

***Keywords :*** Combinatorial optimization, environment and climate change, landscape connectivity, network flow, shortest path, mixed integer linear programming.

## 1 Introduction

Habitat loss is a major cause of the rapid decline of biodiversity [6]. Besides reducing the available resources, it also increases the habitat fragmentation, i.e., the discontinuities among small habitat areas (also called *patches*) [12]. While habitat loss tends to reduce the size of populations of animals and plants, habitat fragmentation hinders circulation of organisms around the landscapes, by decreasing the access to resources and the gene flow among populations. The extent to which the landscape

facilitates the movement of organisms between habitat patches, also called *Landscape connectivity* [26], then becomes of major importance for biodiversity and its conservation. Accounting for landscape connectivity in restoration or conservation plans thus appears as a key solution to maximize the return on investment of the scarce financial support devoted to biodiversity conservation.

Graph theory proves useful to model habitat connectivity [27]. Specifically, a landscape can be viewed as a directed graph in which vertices are the habitat patches and each arc indicates a way for individuals to move from one patch to another (see Figure 1). The weight of a vertex represents its quality – patch area is often used as a surrogate for quality – and the weight of an arc measures the ease with which an individual can make the corresponding travel. This quantity is often approximated by a decreasing function of the distance between the patches [24]. Interestingly, this approach can be used for a variety of ecological systems, like terrestrial (patches of forests in an agricultural area, networks of lakes or wetlands), riverine (patches are segments of river than can be separated by human constructions like dams that prevent fishes' movement), or marine (patches can be reefs that are connected by flows of larvae transported by currents). Based on this formalism, ecologists proposed many connectivity indicators that aim to quantify the quality of a landscape with respect to the connections between its habitat patches [14, 16, 24].

### 1.1 Indicators of landscape connectivity

From among the proposed indicators, the *Probability of Connectivity* (PC) [24] and its derivative, the *Equivalent Connected Area* (ECA) [23], have received encouraging empirical support [3,18,25]. Given a graph $G = (V, A)$ with weights on edges $(\pi_a)_{a\in A}$ and on vertices $(w_v)_{v\in V}$ , the ECA is defined as

$$ECA(G) = \sqrt{\sum_{s\in V}\sum_{t\in V}(w_s w_t \Pi_{st})}$$

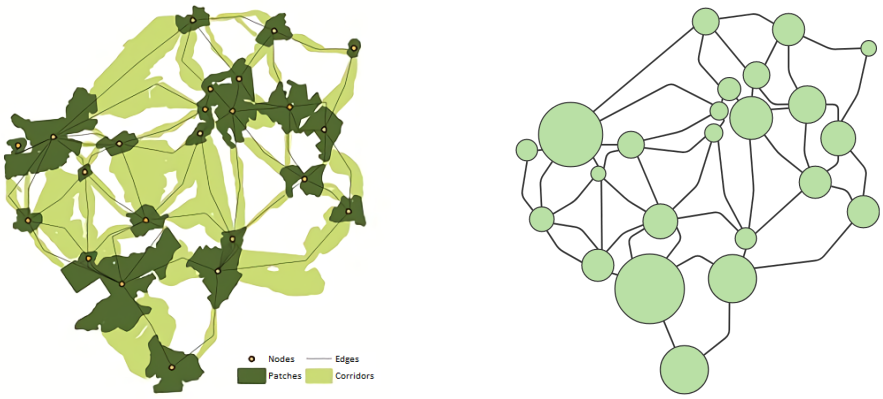
where $\Pi_{st}$ is the *probability of connection* from patch $s$ to patch $t$

$$\Pi_{st} = \max_{P:st\text{-path}} \prod_{a\in P} \pi_a \,.$$

A *most reliable st-path* is a $st$-path that maximizes $\Pi_{st}$ . The rationale underlying the ECA can be summarized as follows. Let $\mathcal{W}$ denote the area of a rectangle containing the landscape under study. Consider a stochastic process that consists in choosing two points $p$ and $q$ uniformly at random in the rectangle. Let PC denote the expected value of a random variable equal to 0 if either $p$ or $q$ does not belong to a patch and $\Pi_{st}$ if $p$ belongs to $s$ and $q$ belongs to $t$ (recall that $\Pi_{st} = 1$ if $s = t$).

Let $w_u$ denote the area of the patch $u$. Since the probability that $p$ belongs to $u$ is $w_u/\mathcal{W}$ and the events $p \in s$ and $q \in t$ are independent, by linearity of expectation, $PC$ can be expressed as follows:

$$PC(G) = \frac{\sum_{s,t \in V} w_s w_t \Pi_{st}}{\mathcal{W}^2} = \frac{ECA(G)^2}{\mathcal{W}^2}.$$




(a) ecological landscape [21] (b) graph representation

Figure 1: Graph representation of an ecological landscape.

The ECA of a landscape is the area of a single patch whose PC value equal to the PC value of the original landscape. If the area of the patches and the landscape are normalized to make $\mathcal{W}$ equal to 1 then PC is the square of ECA. Thus, the problem of optimizing PC is equivalent to the problem of optimizing ECA. The use of ECA is sometimes preferred by researchers interested in landscape connectivity because it directly refers to an area rather than to an expected value of a random variable.

## 1.2 Optimizing ECA under a budget constraint

One key question for which landscape connectivity indicators have been used in the last years is to identify which elements of the landscape (habitat patches, corridors) should be preserved from destruction or restored in order to maintain a well-connected network of habitat under a given budget constraint [2]. Answering this question is equivalent to identifying the set of vertices or arcs that optimally maintains a good level of connectivity. Several mathematical programming models have been introduced in the literature to help decision-makers protect biodiversity. For a review of these models, we refer to the monography [5], the survey article [4] and the references therein. Here, we consider the *Budget-Constrained ECA Optimization problem* (BC-ECA-OPT) which is formally defined in Section 2. This problem takes

as input a landscape with a set of possible improvements $\Phi$. An *arc improvement* consists in increasing the weight of an arc, i.e., the probability of connection between the two patches joined by this arc. In Section 2.2, we consider a version with vertex improvements. A *vertex improvement* consists in increasing the weight of a vertex, i.e., the area/quality of the corresponding patch. Each improvement has a cost and the problem BC-ECA-OPT is to find a combination of improvements that does not exceed a given budget limit, so that the ECA of the resulting landscape is as large as possible. Concrete instances of this problem are described in Section 5.1.

Interestingly, this problem addresses both restoration and conservation cases. The restoration case starts from a currently degraded landscape and aims to restore (e.g., improve quality of patch or facility to use a link, create a new patch) a set of elements among the different feasible options. The conservation case starts with a landscape in which we search for the elements to protect among those that will be altered or destroyed if nothing is done. In the following, we refer to a feasible solution of BC-ECA-OPT as a *restoration/conservation plan*, i.e., a subset of arcs or vertices to be improved or preserved whose total cost does not exceed the budget limit.

### 1.3 Previous works

So far, ecologists mostly tackled this problem by ranking each conservation or restoration option by its independent contribution to ECA, i.e., the amount by which ECA varies if the option is selected alone. Such an approach overlooks the cumulative effects of the decisions made like unnecessary redundancies or potential synergistic effects, e.g., improving two consecutive corridors results in a greater increase in ECA than the sum of the increases achieved by improving each corridor independently. This could lead to solutions that are more expensive or less beneficial to ECA than an optimal solution. Some studies tried to overcome this limitation by considering tuples of options [17,20]. In [20], the authors showed that the brute force approach rapidly becomes impractical for landscapes with more than 20 patches of habitat. Few studies explored the search for an optimal solution but, in most cases, the underlying graph was acyclic (river dendritic networks). For instance, a polynomial time approximation scheme has been proposed in [28] when the underlying graph is a tree. This article describes a dynamical programming algorithm with rounding that computes, for any real $\epsilon > 0$ , a $(1 - \epsilon)$-approximate solution in time $O(n^8/\epsilon)$ where $n$ is the number of vertices of the tree. More recently, [29] introduced a sampling method based on a mixed integer formulation. To our knowledge, this is the only linear programming formulation of BC-ECA-OPT described in the literature. Solving optimally the problem with their mixed integer formulation does not scale to landscapes with few hundreds of patches.

Computing $\Pi_{st}$ for every pair of vertices $s, t \in V$ is a necessary task to compute the $ECA(G)$. If we consider the length function $l$ defined by $l_a = -\log \pi_a$ for $a \in A$ then the distance function $d$ induced by $l$ on $G$ satisfies $d(s,t) = -\log(\Pi_{st})$ for $s, t \in V$. Thus, our problem is closely related to solving the shortest path problem on a graph with different length functions, namely one length function for each restoration plan (or *scenario*). *Mixed Integer Linear Programming* (MILP) can be used to solve such problems; however, the size of MILP formulations is often too large to be handled directly by state-of-the-art MILP solvers. A preprocessing proposed by Catanzaro et al. [7] can be applied to address this issue. It consists in identifying a subset of arcs that can either be removed or contracted to reduce the size of the graph considered. By using the Catanzaro et al. formalism [7], we call an arc $(u, v)$ *t-strong* if it belongs to a shortest path from $u$ to $t$ in every possible scenario. Symmetrically, we call an arc $(u, v)$ *t-useless* if there is no scenario such that $(u, v)$ is on a shortest path from $u$ to $t$. Useless arcs were called 0-persistent in [7]. Since we need to specify a target vertex $t$ for which $(u, v)$ is useless, we didn't adopt the same terminology. In [7], given an arc $(u, v)$ and a vertex $t$, the authors give a sufficient (but not necessary) condition for $(u, v)$ to be $t$-strong and use it to design an $O(|A| + |V| \log |V|)$ time algorithm that decides, in most cases, whether $(u, v)$ is $t$-strong. The authors also proposed a $O(|A| + |V| \log |V|)$ time algorithm to test whether a given arc $(u, v)$ is $t$-useless. In Section 3, we show how the knowledge of $t$-strong and $t$-useless arcs can be used to reduce the size of the MILP formulation and to solve larger instances of BC-ECA-OPT.

### 1.4 Contribution

The contribution of this article is twofold. First, we propose a new MILP formulation which is more compact than the one proposed in [29]. Then, we design a new preprocessing algorithm that computes $t$-strong and $t$-useless arcs for all $t \in V$.

Our new MILP formulation is based on a generalized flow formulation (see for instance [1]) instead of a standard network flow formulation. This change leads to two improvements. Firstly, our formulation has a linear objective function in contrast to the model proposed in [29] which is based on a piecewise constant approximation and additional binary variables to handle non-linearities. Secondly, the new formulation aggregates into a single generalized flow the contribution to the connectivity of several source/sink pairs having the same sink whereas the previous model treated every pair separately. More precisely, our model uses $O(|V||A|)$ flow variables with $O(|V|^2)$ constraints, and $|\Phi|$ integer variables, while the model proposed in [29] is characterized by $O(|V|^2|A|)$ flow variables with $O(|V|^3)$ constraints, and $O(|\Phi| + |V|^2)$ integer variables (recall that $\Phi$ is the set of improvable elements).

The other contribution of this article is a preprocessing step that speeds up the

resolution of the problem by reducing the size of the directed graph on which a generalized flow to a particular sink $t$ has to be computed. Our algorithm improves upon the one proposed by [7] in two ways. First, we replace the sufficient condition defined in [7] by a necessary and sufficient condition. This allows us to compute the entire set of $t$-strong arcs instead of a subset of them. Second, we also improve the time complexity by a factor $|V|$ as we only need to run our algorithm on each arc instead of each pair of arc and vertex.

### 1.5 Organization of the article

Section 2 is devoted to the description of our mixed integer formulation of the problem. In Section 3 we first design and analyze an $O(|A| + |V| \log |V|)$ time algorithm that computes the set of all vertices $t$ for which a given arc $(u, v)$ is $t$-strong or the set of all vertices $t$ for which $(u, v)$ is $t$-useless. Then, we explain why the removal of $t$-useless arcs and the contraction of $t$-strong arcs does not modify the objective function for any restoration/conservation plan. In Section 4, we describe several greedy algorithms for BC-ECA-OPT and provide some instances where they compute solutions that are far from optimal. Section 5 compares our optimization approach to simple greedy algorithms in terms of running times and quality of the solutions found on a set of experimental cases. These experiments show that the preprocessing is very effective and that the greedy approach performs quite well on these instances.

## 2 An improved MILP formulation for BC-ECA-OPT

### 2.1 Basic formulation

In this section, we provide a compact MILP formulation of the problem BC-ECA-OPT that can be formally stated as follows:

**Input:** a graph $G = (V, A)$ with weights on vertices $(w_v)_{v \in V}$ and on arcs $(\pi_a)_{a \in A}$ such that $0 \leq \pi_a \leq 1$ for $a \in A$, a subset $\Phi \subseteq A$ of arcs with weights $(\pi'_a)_{a \in \Phi}$ such that $\pi_a < \pi'_a \leq 1$ for all $a \in \Phi$ and costs $(c_a)_{a \in \Phi}$, and a budget $B \in \mathbb{N}$.

**Output:** A subset $S \subseteq \Phi$ such that $\sum_{a \in S} c_a \leq B$ maximizing $ECA(G(S))$ where $G(S)$ is the graph obtained from $G$ by replacing $\pi_a$ by $\pi'_a$ for all arc $a \in S$.

Our MILP formulation is presented in two steps. First, we decompose $ECA(G)$ into $\sum_{t \in V} w_t f_t$ where $f_t = \sum_{s \in V} w_s \cdot \Pi_{st}$ and explain how $f_t$ can be expressed as the optimal value of a linear programming formulation of a generalized flow problem. Then, we use this result to provide a compact MILP formulation of BC-ECA-OPT.

Recall that a generalized flow differs from a standard network flow by the fact that each arc $a$ has a multiplier $\pi_a$ such that the quantity of flow leaving arc $a$ is equal to the quantity of flow entering in $a$ multiplied by $\pi_a$ (see [1] for an introduction to network flows). The linear program $(\mathcal{P})$ having $f_t$ as optimum value can be defined as follows. Its parameters include the input of the problem BC-ECA-OPT. Namely, the graph $G = (V, A)$ that represents the landscape and the weights $(w_u)_{u \in V}$ and $(\pi_a)_{a \in A}$. The decision variables of $(\mathcal{P})$ are flow variables $(\phi_a)_{a \in A}$ that represent the quantity of flow entering each arc $a \in A$ and a variable $z$ that represents the total quantity of flow sent to $t$. The objective of $(\mathcal{P})$ is to maximize the value of the variable $z$. For each vertex $u \in V$, let $\delta_u^{\text{out}}$ be the set of arcs leaving $u$ and $\delta_u^{\text{in}}$ the set of arcs entering $u$.

$$(\mathcal{P}) \begin{cases} \max & z & & \\ \text{s.t.} & \sum\limits_{a \in \delta_u^{\text{out}}} \phi_a - \sum\limits_{b \in \delta_u^{\text{in}}} \pi_b \cdot \phi_b \leq w_u & u \in V \setminus \{t\} & \text{(A1)} \\ & \sum\limits_{a \in \delta_t^{\text{out}}} \phi_a - \sum\limits_{b \in \delta_t^{\text{in}}} \pi_b \cdot \phi_b = w_t - z & & \text{(A2)} \\ & \phi_a \geq 0 & a \in A & \text{(A3)} \end{cases}$$

Constraints (A1) require that the total quantity of flow leaving $u$ is at most $w_u$ plus the total quantity of flow entering $u$, i.e., the quantity of flow available in vertex $u$. Constraint (A2) requires that $z$ is equal to the total quantity of flow entering $t$ plus $w_t$ minus the total quantity of flow leaving $t$. Finally, constraints (A3) state that each arc carries a non-negative quantity of flow.

**Lemma 1.** *For a fixed vertex $t \in V$, any optimal solution of $(\mathcal{P})$ is obtained by sending, for every vertex $s \in V - \{t\}$, $w_s$ units of flow from $s$ to $t$ along a most reliable $st$-path, i.e., an $st$-path $P$ maximizing $\prod_{a \in P} \pi_a$.*

*Proof.* First notice that when $w_s$ units of flow are sent along an $st$-path $P$ the quantity of flow arriving at $t$ is $w_s$ times the probability $\prod_{a \in P} \pi_a$ of path $P$. Let $\phi'$ be an optimal solution of $(\mathcal{P})$ that maximizes the quantity of flow routed along a path which is not a most reliable path. Suppose, by contradiction, that $\phi'$ sends $\epsilon > 0$ units of flow along an $st$-path $P'$ whose probability is smaller than the probability of a most reliable $st$-path $P$. Let $\phi$ be the flow obtained from $\phi'$ by decreasing the flow sent on path $P'$ by $\epsilon$ and increasing the flow sent on path $P$ by $\epsilon$. Since the probability of $P$ is larger than the probability of $P'$, $\phi$ sends more flow to $t$ than $\phi'$, leading to a contradiction with the choice of $\phi'$. □

**Corollary 1.** *The optimal value of $(\mathcal{P})$ is $f_t = \sum_{s \in V} w_s \cdot \Pi_{st}$.*

*Proof.* Since the objective is to maximize $z$, no flow leaves $t$ in any optimal solution

of $(\mathcal{P})$, i.e., $\sum_{a \in \delta_t^{\text{out}}} \phi_a = 0$. Constraint (A2) ensures that $z$ is the quantity of flow received by $t$ plus $w_t$. By Lemma 1, there exists an optimal solution $\phi$ such that every vertex $s$ distinct from $t$ sends $w_s$ units of flow on a most reliable $st$-path. Hence, for every vertex $s$ distinct from $t$, the flow received by $t$ from $s$ is $w_s \cdot \Pi_{st}$ and thus the value of $z$ is $\sum_{s \in V} w_s \cdot \Pi_{st}$. □

In the light of the above notations, definitions and results, we can now present our MILP program for BC-ECA-OPT. Let $G'$ be the graph obtained from $G$ by adding, for each arc $a = (u, v) \in \Phi$, another arc $a' = (u, v)$ of weight $\pi_{a'} = \pi'_a$. We denote by $\Psi$ the set of copies of arcs in $\Phi$. In the following MILP program, the set of arcs $\delta_u^{\text{out}}$ and $\delta_u^{\text{in}}$ are defined with respect to $G' = (V, A')$ where $A' = A \cup \Psi$. The arc $a'$ is a copy of the arc $a$ but with a weight $\pi'_a$ larger than $\pi_a$, i.e., an improved copy of the arc $a$ in the sense that the probability of successfully moving along the arc $a'$ is greater than that of the arc $a$. Our model ensures that this improved copy of arc $a$ can carry flow only if it is selected in the solution $S$ of BC-ECA-OPT. Indeed, for each arc $a \in \Phi$, we introduce a binary decision variable $x_a$ which is equal to 1 if $a$ is selected in $S$ and 0 otherwise. The BC-ECA-OPT linear programming formulation includes one copy of $(\mathcal{P})$ for each sink $t \in V$. Specifically, we introduce a decision variable $f_t$ that represents the value of $\sum_{s \in V} w_s \cdot \Pi_{st}$ for each sink $t \in V$ and a flow variable $\phi_a^t$ that represents the quantity of flow going through arc $a$ towards $t$, for every sink $t \in V$ and every arc $a \in A'$. The objective of the linear program BC-ECA-OPT is to maximize $\sum_{t \in V} w_t \cdot f_t$, i.e., to find a subset of arcs $S \in \Phi$ such that the value of $ECA(G(S))$ is maximum.

$$
\text{BC-ECA-OPT} \begin{cases}
\max & \displaystyle\sum_{t \in V} w_t \cdot f_t & & \\
\text{s.t.} & \displaystyle\sum_{a \in \delta_u^{\text{out}}} \phi_a^t - \sum_{b \in \delta_u^{\text{in}}} \pi_b \cdot \phi_b^t \leq w_u & t \in V,\ u \in V \setminus \{t\} & \text{(B1)} \\
& \displaystyle\sum_{a \in \delta_t^{\text{out}}} \phi_a^t - \sum_{b \in \delta_t^{\text{in}}} \pi_b \cdot \phi_b^t = w_t - f_t & t \in V & \text{(B2)} \\
& \phi_{a'}^t \leq x_a \cdot M_a & t \in V,\ a \in \Phi & \text{(B3)} \\
& \displaystyle\sum_{a \in \Phi} c_a \cdot x_a \leq B & & \text{(B4)} \\
& x_a \in \{0, 1\} & a \in \Phi & \text{(B5)} \\
& \phi_a^t \geq 0 & t \in V,\ a \in A' & \text{(B6)}
\end{cases}
$$

Constraints of (B1) and (B2) are simply constraints of (A1) and (A2) for every sink $t \in V$. For each arc $a \in \Phi$, the big-M constraints (B3) ensure that if $a$ is not selected in $S$, i.e., $x_a = 0$, then the flow on arc $a'$ is null. The constant $M_a$ is an upper bound on the flow value on arc $a$. We could simply take $M_a = \sum_{u \in V} w_u$ for all $a \in \Phi$ but more precise estimations are possible for a better linear programming relaxation

and thus a faster resolution of the MILP program. Constraint (B4) ensures that the total cost of the improvements is at most $B$. This MILP program has $O(|V|(|A| + |\Phi|))$ flow variables, $|\Phi|$ binary variables and $O(|V|^2 + |V||\Phi|)$ constraints.

### 2.2 Extension to patch improvements

Here, we explain how to extend our model to a version with patch improvements where it is possible to increase the weight of a vertex $u$ from $w_u$ to $w_u^+$ at cost $c_u$. To this end, we process all the occurrences of $w_u$ as follows. Let $y_u$ denote a binary variable equal to 1 if the vertex $u$ is improved and 0 otherwise. We add a term $c_u y_u$ in the budget constraint for every vertex $u$ in the set $W$ of vertices that can be improved. When $w_u$ appears as an additive constant, we simply replace it by $w_u + y_u(w_u^+ - w_u)$. Note that $w_u$ only appears as a coefficient in the objective function in the form $w_u f_t$. In this case, to avoid a quadratic term, we use the classical McCormick linearization [13]. Specifically, we replace the product $w_u f_t$ by $w_u f^t + (w_u^+ - w_u) f_t'$ where $f_t'$ is a new variable that is equal to $f_t$ if $y_u = 1$ and 0 otherwise. To achieve this values of $f_t'$, for all $u \in V$, we add the constraints $f_t' \leq f_t$ and $f_t' \leq y_u M$ where $M$ is larger than any values of $f_t$. As it is a maximization program and $f_t'$ appears with a positive coefficient in the objective function, the first constraint guarantees that $f_t' = f_t$ if $y_u = 1$ in any optimal solution and the second constraint guarantees that $f_t' = 0$ if $y_u = 0$.

## 3 Preprocessing

The size of the mixed integer programming formulation of BC-ECA-OPT presented in Section 2 grows quadratically with the size of the graph that represents the landscape. In this section, we describe a preprocessing step that reduces the size of this graph. To this end, we adopt the approach used by Catanzaro et al. [7] for the Robust Shortest Path Problem. We introduce a notion of *strongness* of an arc with respect to a target vertex $t \in V$. We call a solution $x \in \{0,1\}^\Phi$ of BC-ECA-OPT a *scenario* and say that the distances are *computed under scenario* $x$ when the length of every $a \in A$ is $l_a^-$ if $x_a = 1$ and $l_a$ otherwise. We denote by $d_x(s,t)$ the distance between $s$ and $t$ when the arc lengths are set according to the scenario $x$. As explained in Section 1, recall that an arc $(u,v)$ is *$t$-strong* if, for every scenario $x \in \{0,1\}^\Phi$, $(u,v)$ belongs to a shortest path from $u$ to $t$, i.e., $d_x(u,t) = d_x(u,v) + d_x(v,t)$ for every $x \in \{0,1\}^\Phi$. An arc $(u,v)$ is said to be *$t$-useless* if, for every scenario $x \in \{0,1\}^\Phi$, arc $(u,v)$ does not belong to any shortest $ut$-path when arc lengths are set according to the scenario $x$. Useless arcs were called 0-persistent in [7] but here we need to specify the target $t$. We denote by $S(t)$ the set of arcs $(u,v)$ such that $(u,v)$ is $t$-strong and by $W(t)$ the set of arcs $(u,v)$ such

that $(u, v)$ is $t$-useless. In [7], given an arc $(u, v)$ and a vertex $t$, the authors identify a sufficient (but not necessary) condition for $(u, v) \in S(t)$ and use it to design an $O(|A| + |V| \log |V|)$ time algorithm that can, in most cases, identify if $(u, v) \in S(t)$. The authors also propose an $O(|A| + |V| \log |V|)$ time algorithm to test whether a given arc $(u, v)$ belongs to $W(s)$ or not. In Section 3.1, given an arc $(u, v)$ of $G$, we show how to adapt Dijkstra's algorithm to compute in $O(|A| + |V| \log |V|)$ time the set of all vertices $t$ such that $(u, v) \in S(t)$ or the set of all vertices $t$ such that $(u, v) \in W(t)$. This improves the results of [7] by providing a necessary and sufficient condition for $(u, v)$ to be $t$-strong, i.e., we compute the entire set $S(t)$ while the algorithm of [7] computes a subset of $S(t)$. It also reduces the time complexity for computing $S(t)$ and $W(t)$ for all $t$ by a factor $|V|$ as we only need to run the algorithms on each arc instead of each pair of arc and vertex. Then, we explain how the knowledge of $S(t)$ and $W(t)$ for all $t$ can be used to define a smaller equivalent instance of the problem.

### 3.1 Computing $S(t)$ for all $t$

Given a scenario $x \in \{0, 1\}^\Phi$, the *fiber* $F_x(u, v)$ of arc $(u, v) \in A$ is the set of vertices $t$ such that $(u, v)$ belongs to a shortest path from $u$ to $t$ when arc lengths are set according to $x$, i.e.,

$$F_x(u, v) = \{t \in V : d_x(u, t) = l_x(u, v) + d_x(v, t)\}.$$

Let $F(u, v)$ be the intersection of the fibers of $(u, v) \in A$ over all possible scenarios $x \in \{0, 1\}^\Phi$, i.e., $F(u, v) := \bigcap_x F_x(u, v)$. By definition, an arc $(u, v)$ is $t$-strong if $t$ belongs to $F_x(u, v)$ for every scenario $x$; i.e.,

$$S(t) = \{(u, v) : t \in F(u, v)\}$$

In order to compute every $S(t)$, we first compute $F(u, v)$ for every arc $(u, v) \in A$ and then we transpose the representation to get $S(t)$ for every vertex $t \in V$. Let $y \in \{0, 1\}^\Phi$ be the following scenario :

$$y_{wt} = \begin{cases} 0 & w \in F(u, v) \textbf{ or } (w, t) = (u, v) \\ 1 & w \notin F(u, v) \end{cases} \tag{1}$$

**Lemma 2.** *The intersection $F(u, v)$ of the fibers of $(u, v) \in A$ over all possible scenarios is the fiber of $(u, v)$ under the scenario $y$, i.e., $F(u, v) = F_y(u, v)$.*

*Proof.* The inclusion $F(u, v) \subseteq F_y(u, v)$ is trivial. Thus, it suffices to prove that $F_y(u, v) \subseteq F_x(u, v)$ for any scenario $x \in \{0, 1\}^\Phi$. By contradiction, suppose that $x$ is a scenario such that $F_y(u, v) \setminus F_x(u, v)$ contains a vertex $t$ at minimum distance

from $u$ in the scenario $y$. Let $P$ denote a shortest $ut$-path containing $(u,v)$ under the scenario $y$. For every vertex $w$ of $P$, $d_y(u,w) \leq d_y(u,t)$ and $w \in F_y(u,v)$. Hence, by the choice of $t$, $w$ belongs to $F_x(u,v)$ for every scenario $x$. Therefore, $w$ belongs to $F(u,v)$ and the length of every arc of $P$ is set to its upper bound in the scenario $y$, i.e., $l_y(P) = l(P)$. Now, let $Q$ be a shortest $ut$-path in the scenario $x$. If the path $Q$ contains a vertex $z \in F(u,v)$ then $d_x(u,t) = d_x(u,z) + d_x(z,t) = l_x(u,v) + d_x(v,z) + d_x(z,t) = l_x(u,v) + d_x(v,t)$, a contradiction with $t \notin F_x(u,v)$. Therefore, the vertices of $Q$ do not belong to $F(u,v)$ and thus their lengths are set to their lower bound in $y$, i.e., $l(Q) = l^-(Q)$. We deduce that $l_x(P) \leq l_y(P) \leq l_y(Q) \leq l_x(Q)$, which contradicts $t \notin F_x(u,v)$. □

We propose an $O(|A| + |V|\log|V|)$ time algorithm that, given an arc $(u,v)$, computes simultaneously the scenario $y$ defined by (1) and the fiber $F_y(u,v)$ of arc $(u,v)$ with respect to this scenario. The algorithm is an adaptation of the Dijkstra's shortest path algorithm that assigns colors to vertices. We prove that it colors a vertex $w$ in blue if $w$ belongs to $F_y(u,v)$ and in red otherwise. At each step, we consider a subset of vertices $S \subseteq V$ whose colors have been already computed. Before the first iteration, $S = \{u\}$ and the length of every arc leaving $u$ is set to its lower bound except the length of $(u,v)$ which is set to its upper bound according to scenario $y$. At each step, since the color of every vertex of $S$ is known, the length, under scenario $y$, of every arc $(w,t)$ with $w \in S$ is also known. Therefore, it is possible to find a vertex $t \in V - S$ at minimum distance from $u$, under scenario $y$, in the subgraph $G[S \cup \{t\}]$ induced by $S \cup \{t\}$. Following Dijkstra's algorithm analysis, we know that the distance under scenario $y$ from $u$ to $t$ in $G[S \cup \{t\}]$ is in fact the distance between $u$ and $t$ in $G$. This allows us to determine the existence of a shortest path from $u$ to $t$ in the scenario $y$ passing via $(u,v)$ and to color the vertex $t$ accordingly. We conclude the iteration with a new vertex $t$ whose color is known and that can be added to $S$ before starting the next iteration. Since the algorithm adds a new vertex to $S$ at the end of each loop, it halts after $|V| - 1$ iterations.

**Algorithm 1:** Computes the set $S$ of vertices $t$ such that $(u, v)$ is $t$-strong

**Input :** $G = (V, A, l, l^-)$, $(u, v) \in A$
**Output:** $\{t \in V : (u, v) \text{ is } t\text{-strong}\}$
**foreach** $(u, w) \in \delta_u^{out} \setminus \{(u, v)\}$ **do**
  $d(w) \leftarrow l^-_{uw}$
  $\gamma(w) \leftarrow red$
$d(v) \leftarrow l_{uv}$; $\gamma(v) \leftarrow blue$
$S \leftarrow \{u\}$; $\gamma(u) \leftarrow red$
**while** $S \neq V$ **do**
  Pick $t \in V - S$ with smallest $d(t)$ breaking ties by choosing a vertex $t$ such that $\gamma(t) = blue$ if it exists
  **if** $\gamma(t)$ *is blue* **then**
    **foreach** $(t, w) \in \delta_t^{out}$ *such that* $d(w) \geq d(t) + l_{tw}$ **do**
      $d(w) \leftarrow d(t) + l_{tw}$
      $\gamma(w) \leftarrow blue$
  **else**
    **foreach** $(t, w) \in \delta_t^{out}$ *such that* $d(w) > d(t) + l^-_{tw}$ **do**
      $d(w) \leftarrow d(t) + l^-_{tw}$
      $\gamma(w) \leftarrow red$
  $S \leftarrow S \cup \{t\}$
**return** $\{t \in V : \gamma(t) = blue\}$

For every vertex $w \in V - S$, the estimated distance $d(w)$ is the length of a $uw$-path under the scenario $y$ in the subgraph $G[S \cup \{w\}]$. An arc $a$ is blue if $a = uv$ or if its origin is blue, the other arcs are red, i.e., $a$ is blue if $y_a = 0$ and red if $y_a = 1$. The estimated color $\gamma(w)$ of $w$ is blue if there exists a blue $uw$-path of length $d(w)$ in $G[S \cup \{w\}]$ and red otherwise. The correctness of Algorithm 1 follows from the following lemma.

**Lemma 3.** *For every vertex $t \in S$, $\gamma(t)$ is blue if and only if $t \in F_y(u, v)$.*

*Proof.* We proceed by induction on the number of vertices of $S$. When $S = \{u\}$, the property is verified. Now, suppose the property holds before the insertion in $S$ of the vertex $t$ such that $d(t)$ is minimum. By the induction hypothesis, the lengths of arcs having their source in $S$ are set according to $y$. Therefore, following Dijkstra's algorithm analysis, we deduce that $d(t)$ is the length of the shortest path from $u$ to $t$ in the graph $G$ under scenario $y$. If $t$ has been colored in blue then there exists a blue vertex $w \in S$ such that $d(t) = d(w) + l_{wt}$. By the induction hypothesis $w \in F_y(u, v)$ and there exists a shortest $ut$-path under scenario $y$ containing $(u, v)$, i.e., $t \in F_y(u, v)$. Now, suppose that $t$ has been colored in red. By contradiction, assume there exists a shortest $ut$-path under scenario $y$ that contains $(u, v)$. This

path cannot contain a vertex outside $S$ except $t$ because otherwise, since arc lengths are non-negative, its length according to $y$ would be greater than $d(t)$ by the choice of $t$. Therefore, the predecessor $w$ of $t$ in this path belongs to $S$. Since $w$ belongs to a shortest $ut$-path passing via $(u, v)$, by induction, it was colored in blue. But in this case, there exists a blue vertex $w$ such that $d(w) + l(w, t) = d(t)$, and $t$ was colored in blue as well, leading to a contradiction. □

We are now ready to state the main result of this section.

**Proposition 1.** *Given a graph $G = (V, A)$, two arc-length functions $l^-$ and $l$ such that $0 \leq l^-_a \leq l_a$ for every arc $a \in A$, and an arc $(u, v)$, Algorithm 1 computes in $O(|A| + |V| \log |V|)$ time the set of vertices $t$ such that $(u, v)$ is $t$-strong.*

*Proof.* Lemma 2 shows that given an arc $(u, v)$ the set of vertices $S$ such that $t \in S$ if and only if $(u, v)$ is $t$-strong is a fiber for a specific scenario. Lemma 3 shows that the algorithm computes this fiber. Therefore, the Algorithm 1 is correct. As for Dijkstra's algorithm, the computational complexity of Algorithm 1 can be reduced to $O(|A| + |V| \log |V|)$ by using a Fibonacci heap as priority queue. □

### 3.2 Computing $W(t)$ for all $t$

Given an arc $(u, v)$, it is possible to modify Algorithm 1 so that it computes the set of vertices $W$ such that $t \in W$ if and only if $(u, v)$ is $t$-useless. Before describing this modification, we explain how the two problems are related. For that, we first introduce a variant of the notion of strongness. We will say that an arc $(u, v)$ is strictly $t$-strong if it belongs to all shortest $ut$-paths for every scenario $x \in \{0, 1\}^\Phi$. Recall that $t$-strongness requires only the existence of a shortest $ut$-path passing via $(u, v)$ for every scenario $x \in \{0, 1\}^\Phi$. Algorithm 1 can be easily adapted to compute for every arc $(u, v)$ the set of vertices $t$ such that $(u, v)$ is strictly $t$-strong. It suffices to change the way the algorithms break ties between a red and a blue path and the choice of $t$ in case of tie. Namely, in the first internal loop, the condition for coloring $w$ in blue becomes $d(w) > d(t) + l_{tw}$ while the condition for coloring $w$ in red in the second internal loop becomes $d(w) \geq d(t) + l^-_{tw}$. Moreover, when we choose $t$ such that $d(t)$ is minimal, we break ties by choosing a red vertex if it exists. Clearly, these small changes exclude the existence of a red path of length $d(w)$ between $u$ and a blue vertex $w$. Therefore, $(u, v)$ belongs to every path of length $d(w)$ and $(u, v)$ is strictly $w$-strong whenever $w$ is blue. We call the resulting algorithm the strict version of Algorithm 1. The next step is to extend the notion of strict strongness to a subset of arcs having the same source. For any vertex $u \in V$, a subset $\Gamma \subseteq \delta^{\text{out}}_u$ of arcs is strictly $t$-strong if, for every scenario $x \in \{0, 1\}^\Phi$, all shortest $ut$-paths intersect

**Algorithm 2:** Computes the set of vertex $t$ such that $(u, v)$ is $t$-useless

**Input** : $G = (V, A, l, l^-)$, $(u, v) \in A$
**Output:** $\{t \in V : (u, v)$ is $t$-useless$\}$
$d(v) \leftarrow l^-_{uv}$; $\gamma(v) \leftarrow red$
**foreach** $(u, w) \in \delta^{out}_u \setminus \{(u, v)\}$ **do**
  $d(w) \leftarrow l_{uw}$
  $\gamma(w) \leftarrow blue$
$S \leftarrow \{u\}$; $\gamma(u) \leftarrow blue$
**while** $S \neq V$ **do**
  Pick $t \in V - S$ with smallest $d(t)$ breaking ties by choosing a vertex $t$ such that $\gamma(t) = red$ if it exists
  **if** $\gamma(t)$ *is blue* **then**
    **foreach** $(t, w) \in \delta^{out}_t$ *such that* $d(w) > d(t) + l_{tw}$ **do**
      $d(w) \leftarrow d(t) + l_{tw}$
      $\gamma(w) \leftarrow blue$
  **else**
    **foreach** $(t, w) \in \delta^{out}_t$ *such that* $d(w) \geq d(t) + l^-_{tw}$ **do**
      $d(w) \leftarrow d(t) + l^-_{tw}$
      $\gamma(w) \leftarrow red$
  $S \leftarrow S \cup \{t\}$
**return** $\{t \in V : \gamma(t) = blue\}$

$\Gamma$. By definition, an arc $(u, v)$ is $t$-useless if and only if $\delta^{\text{out}}_u \setminus \{(u, v)\}$ is strictly $t$-strong. Indeed, every $ut$-path avoiding $(u, v)$ intersects $\delta^{\text{out}}_u \setminus \{(u, v)\}$ and, conversely, $(u, v)$ belongs to every $ut$-path avoiding $\delta^{\text{out}}_u \setminus \{(u, v)\}$. Hence, computing the set of vertices $t$ such that $(u, v)$ is $t$-useless amounts to computing the set of vertices $t$ such that $\delta^{\text{out}}_u \setminus \{(u, v)\}$ is strictly $t$-strong. An algorithm that computes this set of vertices can be obtained from the strict version of Algorithm 1 by modifying only the initialization step: arc $(u, v)$ is colored in red and its length is set to $l^-_{uv}$ while arcs of $\delta^{\text{out}}_u \setminus \{(u, v)\}$ are colored in blue and their lengths are set to their upper bound. A correctness proof very similar to the one of Algorithm 1 (and that we will not repeat) shows that a vertex is colored blue by Algorithm 2 if and only if $(u, v)$ is $t$-useless. Since the two algorithms have clearly the same time complexity, we conclude this section with the following result:

**Proposition 2.** *Given a graph $G = (V, A)$, two arc-length functions $l^-$ and $l$ such that $0 \leq l^-_a \leq l_a$ for every arc $a \in A$, and an arc $(u, v)$, Algorithm 2 computes in $O(|A| + |V| \log |V|)$ time the set of vertices $t$ such that $(u, v)$ is $t$-useless.*

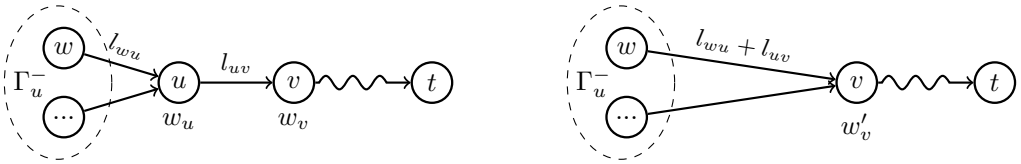
### 3.3 Reducing the size of the graph

**Removal of an arc from** $W(t)$. Consider an arc $(u,v) \in W(t)$. Since $(u,v)$ does not belong to any shortest path from $u$ to $t$ for all scenarios $x \in \{0,1\}^\Phi$, its removal does not affect the distance from any vertex to $t$. From the definition of ECA, the removal of $(u,v)$ clearly does not affect the contribution of $t$ to ECA. Let $f_t^x(G) := \sum_{s \in V} w_s w_t \Pi_{st}^x$ be the contribution to ECA of all the pairs having sink $t$ in the graph $G$ when the probabilities of connection $\Pi_{st}^x$, $s \in V$, are computed under the scenario $x$. The following result holds:

**Lemma 4.** *Let $(u,v) \in W(t)$ and let $G'$ be a graph obtained from $G$ by removing arc $(u,v)$. Then, for all scenario $x \in \{0,1\}^\Phi$, it holds that $f_t^x(G) = f_t^x(G')$.*

*Proof.* For every scenario $x \in \{0,1\}^\Phi$, $(u,v)$ does not belong to any shortest path from $u$ to $t$. Therefore, the removal of $(u,v)$ cannot affect the probability of connection $\Pi_{ut}$ for any vertex $t$, Thus $f_t^x(G) = f_t^x(G')$. □

**Contraction of an arc in** $S(t)$. Consider an arc $(u,v) \notin \Phi$ such that $(u,v) \in S(t)$. The contraction of $(u,v)$ consists in replacing every arc $(w,u) \in \delta_u^{\text{in}}$ by an arc $(w,v)$ of length $l'_{wv} = l_{wu} + l_{uv}$ and by removing the vertex $u$ and all its outgoing arcs. The weight of $u$ in $G$ is moved to the weight of $v$ in $G'$. Namely, the weight of $v$ in the new graph is $w'_v = w_v + w_u e^{-l_{uv}}$. Let $G'$ be the graph obtained from $G$ by contracting $(u,v)$. The next lemma establishes that the contribution of $t$ to ECA in $G$ is equal to its contribution in $G'$.




(a) (b)

Figure 2: (a) A graph $G$ before contraction of an arc $(u,v)$. (b) A graph $G'$ obtained from $G$ by contracting arc $(u,v)$. The weight $w'_v$ of $v$ in $G'$ is equal to $w_v + w_u e^{-l_{uv}}$.

**Lemma 5.** *Let $(u,v) \in S(t)$ and let $G'$ be the graph obtained from $G$ by contracting arc $(u,v)$ and modifying accordingly the weight of $w_v$. For every scenario $x \in \{0,1\}^\Phi$, $f_t^x(G) = f_t^x(G')$.*

*Proof.* Let $s$ denote a vertex of $G$, $x$ a scenario in $\{0,1\}^\Phi$ and $f_{st}^x(G)$ the contribution to ECA of the pair $st$ with scenario $x \in \{0,1\}^\Phi$ in $G$. If $s \notin \{u,v\}$, it is easy to check that, for any $x \in \{0,1\}^\Phi$ the length of the shortest path from $s$ to $t$ in $G$ is equal to the length of the shortest path from $s$ to $t$ in $G'$. Moreover, the weight of $s$

is the same in $G$ and in $G'$. Therefore, by the definition of ECA, $f^x_{st}(G) = f^x_{st}(G')$, for any $x \in \{0,1\}^\Phi$. On the other hand, the contributions of the pairs $ut$ and $vt$ in $G$ sum to the contribution of $v$ in $G'$. Indeed,

$$\begin{aligned} f^x_{ut}(G) + f^x_{vt}(G) &= w_u e^{-d_x(u,t)} + w_v e^{-d_x(v,t)} \\ &= w_u e^{-(l_{uv}+d_x(v,t))} + w_v e^{-d_x(v,t)} \\ &= (w_v + w_u e^{-l_{uv}}) e^{-d_x(v,t)} \\ &= w'(v) e^{-d_x(v,t)} \\ &= f^x_{vt}(G'). \end{aligned}$$

We conclude that, for any $x \in \{0,1\}^\Phi$,

$$\begin{aligned} f^x_t(G) &= \sum_{s \in V} f^x_{st}(G) \\ &= f^x_{ut}(G) + f^x_{vt}(G) + \sum_{s \in V - \{u,v\}} f^x_{st}(G) \\ &= f^x_{vt}(G') + \sum_{s \in V - \{u,v\}} f^x_{st}(G') \\ &= f^x_t(G'). \end{aligned}$$

□

**Graph reduction.** Let $G_t$ denote the graph obtained from $G$ by deleting every arc of $W(t)$ and contracting every arc of $S(t)$. The preprocessing step consists in computing the graph $G_t$ for every vertex $t$. For that, we compute $F(u,v)$ for every arc $(u,v)$ in $O(|A| \cdot (|A| + |V| \log |V|))$ time. Then we transpose the representation to obtain $S(t)$ for each vertex $t$. Similarly, we compute $W(t)$ for each vertex $t$ within the same time complexity. The experiments of Section 5 show that, when the number of arcs that can be protected is much smaller than the total number of arcs, replacing $G$ by $G_t$ in the construction of the mixed integer program reduces significantly the size of the MILP formulation and the running times.

# 4 Greedy algorithms and their limits

In view of the lack of an efficient method to compute an optimal solution of BC-ECA-OPT, ecologists often use greedy algorithms to compute sub-optimal solutions [11]. In this section, we present four commonly used *greedy algorithms* and highlight their pathological cases. In Section 5.3, we compare the quality of the solutions obtained with these greedy algorithms with the optimal solution on four case studies.

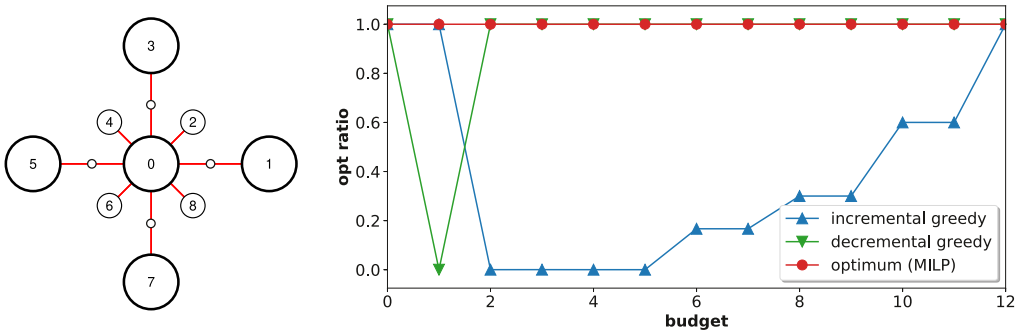
The *Incremental Greedy* (IG) algorithm starts from the graph with no improved

element. At each step $i$, the algorithm selects the element $e$ with the greatest ratio $\delta_e^i/c_e$ until no more element fits in the budget. Here, $\delta_e^i$ denotes the difference between the value of ECA with and without the improvement of the element $e$ at the step $i$. As usual, $c_e$ is the cost of improving the element $e$. The element $e$ can be either an arc or a vertex.

The *Decremental Greedy* (DG) algorithm, similar to the Zonation Algorithm [15], starts from the graph with all improvements performed and iteratively removes the improvement of the element $e$ with the smallest ratio $\delta_e^i/c_e$. DG finishes with incremental steps to ensure there is no free budget left. These algorithms perform at most $|\Phi|$ steps and at each step $i$ need to compute $\delta_e^i$ for each element $e$. It is easy to implement IG in $O(|V|^3 + |\Phi|^2 \cdot |V|^2)$ time by using an all pairs shortest path algorithm to compute in $O(|V|^3)$ time the initial distance matrix and then by performing $|\Phi|$ steps in which the computation of $\delta_e^i$ for each arc $e \in \Phi$ takes $O(|V|^2)$ time. Indeed, when we decrease the length of an edge we can update the distance matrix in $O(|V|^2)$ time. For the implementation of DG, when we increase the length of an edge we cannot update the distance matrix as easily as for IG. We can recompute in $|V|^3$ time the whole distance matrix for computing each $\delta_e^i$ and the algorithm runs in $O(|\Phi|^2 \cdot |V|^3)$ time. Dynamically updating shortest path lengths would improve the computational complexity [8]. These complexities are already too large for the practical instances handled by ecologists which can have few thousands of patches. Most of the studies using the PC or ECA indicators use simpler algorithms that we call *Static Increasing* (SI) and *Static Decreasing* (SD). These algorithms are variants of the greedy algorithms which do not recompute the ratio $\delta_e/c_e$ of each element $e$ at each step and thus are faster but do not account for cumulative effects nor redundancies.

Below, we provide instances on which IG and DG perform poorly compared to an optimal solution. On these instances, it is easy to check that the solutions returned by SI and SD are not better than the solutions returned by IG and DG. In the following instances, all arcs have a probability 1 if improved and 0 otherwise and have unitary costs. Recall that a spider is a tree consisting of several paths glued together on a central vertex (Figure 3a, 4a and 5a).

**Bad case for IG:** The graph is a spider with $2k$ branches: $k$ long branches with two edges, an intermediate vertex of weight 0 and a leaf vertex of weight 1, and $k$ short branches consisting of a single edge with a leaf of very small weight $\epsilon > 0$; see Figure 3a. All branches are connected to a central vertex of weight 1. IG performs poorly on this instance. Indeed, IG is tricked into selecting short branches with very small ECA improvement because selecting an edge of a long branch alone does not increase ECA at all. An optimal solution results in a larger value of ECA by




(a) (b)

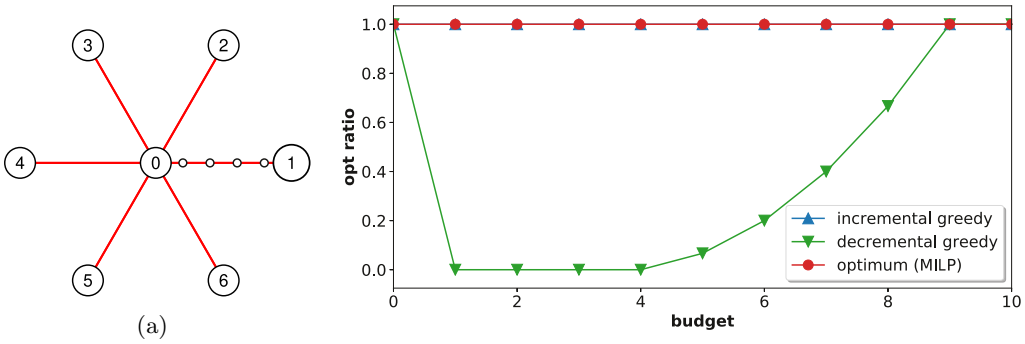
Figure 3: An instance on which Incremental Greedy (IG) fails. (a) the graph of the IG bad case with $k = 4$, (b) ratio of the increase in ECA between the solutions returned by IG and DG and an optimal solution for several budgets.

improving pairs of arcs of long branches; see Figure 3b. For this instance, IG does not perform well while DG finds an optimal solution computed by the MILP solver except when the budget is $1$. In this case, the reverse occurs: DG performs badly while IG is optimal. Indeed, DG realizes that the budget is not sufficient to improve two arcs of a long branch only after removing the improvements of all short branches.

**Bad case for DG:** The graph is obtained from a star with $k + 1$ branches by replacing one branch by a path of length $k$; see Figure 4a. The central vertex and all leaves except the leaf of the path have weight $1$. The leaf of the path has weight $1 + \epsilon$. The internal vertices of the path have weight $0$. DG performs poorly on this instance because it removes one by one the branches of the star for which $\delta_e/c_e = 1$ before removing an edge of the path for which $\delta_e/c_e = 1 + \epsilon$. When the budget is at least $2$, an optimal solution removes all the edges of the path before removing an edge of another branch; see Figure 4b.

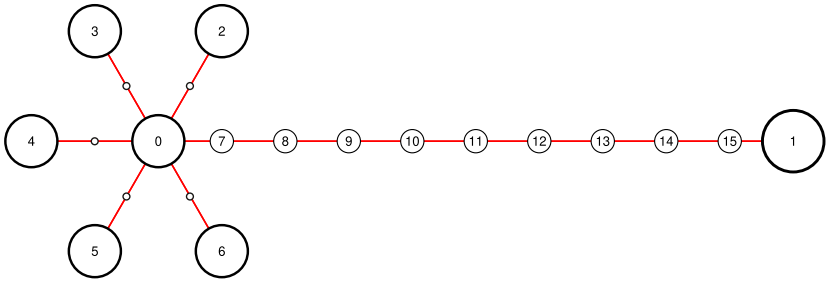
**Bad case for IG and DG:** The graph is a spider with $k+1$ branches. All branches except one are paths of length 2 with an internal vertex of weight 0 and a leaf of weight $1$. The last branch is a path of length $2k$ with internal vertices of weight $\epsilon > 0$ and a leaf of weight $1 + \epsilon$. All branches intersect in a central vertex of weight 1; see Figure 5a. In this case, both IG and DG fail. On the one hand, IG selects the edges of the path of length $2k$ one by one and does not realize that by taking two edges of a short branch it could improve ECA much more. On the other hand, DG removes first the edges of the short branch because the weight of leaf of a long branch is $1 + \epsilon$ while the weight of the leaf of a short branch is $1$. Hence, DG and IG return the same low quality solution.

The case of Figure 5 illustrates the fact that IG and DG do not provide any

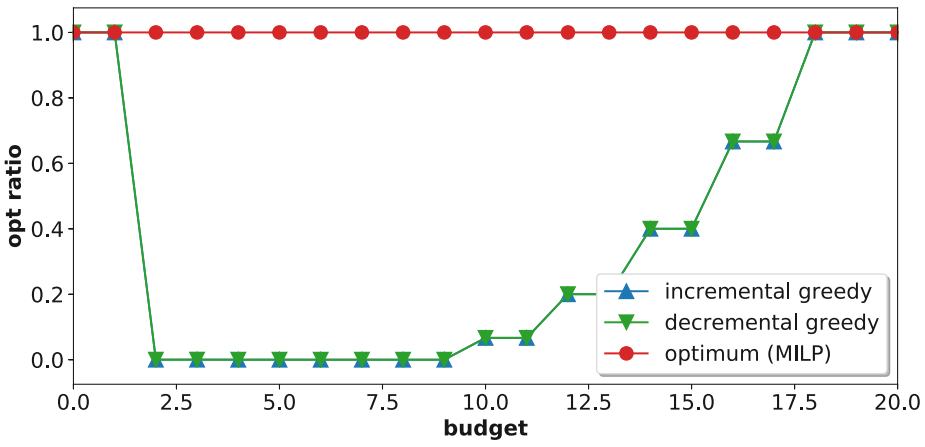



(a) (b)

Figure 4: An instance on which Decremental Greedy (DG) fails. (a) the graph of the DG bad case with $k = 5$, (b) ratio of the increase in ECA between the solutions returned by IG and DG and an optimal solution for several budgets.




(a)




(b)

Figure 5: An instance on which both Incremental and Decremental Greedy algorithms fail. (a) the graph of the IG and DG bad case with $k = 5$, (b) ratio of the increase in ECA between the solutions returned by IG and DG and an optimal solution for several budgets.

constant approximation guarantee (even for trees), i.e., for any constant $0 < \alpha < 1$ there exists an instance of BC-ECA-OPT such that $ALG < \alpha OPT$ where ALG is the value of ECA for the best solution among those returned by IG and DG and $OPT$ is the value of ECA for an optimal solution.

# 5 Numerical experiments

In this section, we report on our computational experiments in order to highlight the combined benefit of our MILP formulation and preprocessing step. We performed the numerical experiments on a desktop computer equipped with an Intel(R) Core(TM) i7-8700k 4.8 gigahertz and 32 gigabytes of memory and running Manjaro Linux release 21.2.4 with GCC version 10.2 and the libstdc++ that comes with it. We implemented our model as well as the preprocessing and greedy algorithms in C++17 using Gurobi Optimizer [10] version 9.1.1 with default settings for solving MILP formulations, the graph library LEMON [9] version 1.3.1 for managing graph algorithms, and the library TBB [19] version 2020.3 for multithreading the preprocessing and greedy algorithms. Code and data are available at `https://gitlab.lis-lab.fr/francois.hamonic/landscape_opt_networks_submission`.

## 5.1 Instances

Below, we briefly describe the case studies on which we conduct experiments.

**Case study 1** consists of identifying among a set of 15 dams present on the Aude river (France) those that need to be equipped with fish passes in order to restore the river connectivity for trouts [22]. The graph is a tree of 45 vertices with 88 arcs of which 30 represent dams and can be improved by increasing their weight from 0 to 0.8 .

**Case study 2** consists of identifying the remnant forest patches that need to be preserved from deforestation in the Montreal neighborhood (Canada) to guarantee habitat connectivity for the wood frog [2]. The graph is a planar graph of 598 vertices and 989 arcs in which 260 vertices can be improved by increasing their quality and 80 arcs can be improved by increasing their weight from 0 to 1 .

**Case study 3** consists of identifying street sections in which planting trees can improve the connectivity of the urban canopy for the European red squirrel in the city of Aix-en-Provence. The graph is a triangular grid of 6186 vertices and 27818 arcs where 47 street sections, each with an average of 90 arcs, can be improved by increasing the weight of each arc $a$ from $\pi_a$ to $\pi_a^{1/6}$.

**Case study 4** consists of identifying wastelands that need to be preserved from soil artificialization to maintain connectivity among urban parks and the surrounding natural massifs in the city of Marseille for songbirds (e.g., Eurasian blackcap). The graph is a near complete graph of 297 vertices and 25024 arcs, of which 100 represent

wastelands and can be improved by increasing their weight from 0 to 1 .

### 5.2 Scalability and benefits of the preprocessing

In this section we address the scalability of our approach and the added benefits of our preprocessing step. For this purpose we execute our method on about one hundred instances obtained from the four case studies by varying the budget between 0 (meaning no option is selected) and 100% (meaning all the options are selected). More precisely, our goal is to understand how the computing times and the size of the MILP model vary with respect to (i) the budget available, (ii) the number of binary decision variables, (iii) the presence of the preprocessing, (iv) and the percentage of improvable arcs compared to the total number of arcs in the graph.

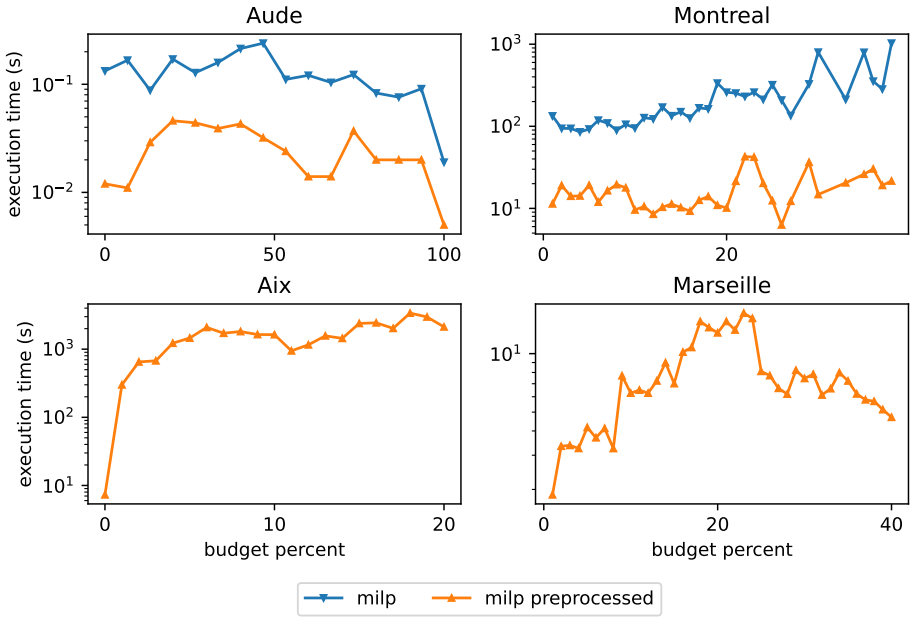



Figure 6: Execution times on the four case studies as a function of the budget (missing points correspond to instances that do not finish within 10 hours)

For the Aude and Montreal cases, the preprocessing reduces the resolution time by about a factor of 10 (Figure 6). Without preprocessing, the Aix and Marseille instances are not solved by the optimizer within 10 hours, whereas with preprocessing they become solvable in about half an hour and half a minute respectively. In most unfinished instances, the optimizer reaches the optimal solution but is not able to complete the exploration of the search space in the allotted time. Some instances of the Montreal case cannot be solved without preprocessing. The average computation times shown in Table 2 do not take these instances into account.

The preprocessing represents a small portion of the total computation time for

| case | MILP | | | Preprocessed MILP | | | |
|---|---|---|---|---|---|---|---|
| | #var | #const | time | #var | #const | p. time | time |
| Aude | 4061 | 2551 | 120 ms | 1069 | 1055 | 3 ms | 20 ms |
| Montreal | 830530 | 262445 | 4 mins | 318848 | 167153 | 0.26 s | 17.26 s |
| Aix | 1748708 | 624841 | – | 555124 | 295010 | 3 s | 1600 s |
| Marseille | 4949825 | 78410 | – | 41676 | 22465 | 0.9 s | 7 s |

Table 1: Comparison of the MILP and the preprocessed MILP according to the number of variables (#var), the number of constraints (#const), the preprocessing time (p. time) and the average computation time of the MILP resolution (time).

| #Wasteland | MILP | | | Preprocessed MILP | | | DG |
|---|---|---|---|---|---|---|---|
| | #var | #const | time | #var | #const | time | time |
| 20 | 345775 | 18410 | 12 s | 10716 | 7197 | $< 1$ s | $< 1$ s |
| 50 | 717901 | 36410 | 2 mins | 34613 | 21485 | 2 s | 7 s |
| 80 | 1306597 | 59810 | 32 mins | 76281 | 42039 | 13 s | 30 s |
| 110 | | - | | 132225 | 66759 | 1 min | 1 min 30 s |
| 140 | | - | | 207999 | 96600 | 3 mins | 3 mins |
| 170 | | - | | 308355 | 132701 | 28 mins | 7 mins |

Table 2: Comparison of the MILP, the preprocessed MILP and DG according to the number of variables (#var), the number of constraints (#const) and the time (on average with 20 different budget values) it takes to solve the Marseille instance with different numbers of wastelands

all case studies (Table 1). The number of variables of the model is reduced by about 75% in the Aude case, 60% in the Montreal case, 70% in the Aix case and 99% in the Marseille case. This last number is explained by the fact that the Marseille graph is near complete and a large proportion of its arcs are $t$-useless for some vertex $t$. Regarding the constraints, the reduction is 60% for the Aude case, 33% for the Montreal one, 53% for the Aix case and about 70% for the Marseille case.

For the case of Marseille, the one with the largest number of variables, we also explored how the number of potential options influences the number of constraints and the computation time. We see in Table 2 that the computation time of the MILP without preprocessing increases very quickly. It takes more than 30 minutes on average for instances with 80+ wastelands whereas the preprocessed ones can be solved in less than 30 minutes with up to 170 wastelands. This is due to the preprocessing step that significantly reduces the time required to solve the linear relaxation by reducing the number of variables, constraints and non-zero entries of the mixed integer program. The preprocessed MILP is faster than the greedy algorithm for instances with at most 140 wastelands (the preprocessing step was not used for greedy algorithms).

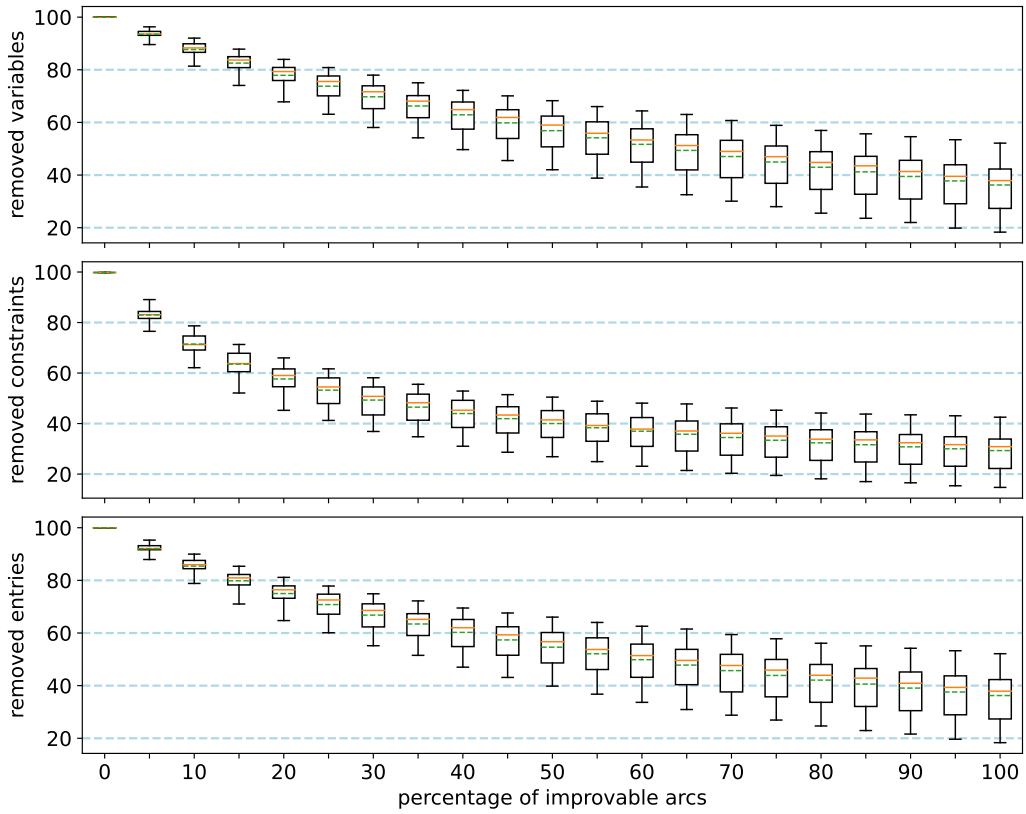



Figure 7: Box plots showing the percentage of constraints, variables and non-zero entries removed by the preprocessing as a function of the percentage of improvable arcs. The red line is the median, the dashed green line is the mean, the box represents the values between the 25th and 75th percentiles and the whiskers the min and max values.

To show the efficiency of the preprocessing on the number of constraints in the problems (which is related to computation time), we run our preprocessing on 400 randomly generated instances from a model of the landscape around Montreal for hares of 8733 vertices and 18422 arcs [2] and study the impact of the preprocessing on the MILP formulation size. For building these instances, we take 20 connected subgraphs of 500 vertices and for each graph we create 20 instances by randomly picking a percentage of arcs whose probability could be increased from $\pi$ to $\sqrt{\pi}$. Sampling in the very large graph representing the Montreal region gives rise to different types of instances in terms of shapes, arc density, etc. This is why we decided to use the Montreal case for these experiments.

Figure 7 shows a box plot of the size reduction of the MILP formulation in terms of constraints, variables and non-zero entries with respect to the percentage of arcs that could be improved compared to the total number of arcs in the graph. The preprocessing removes almost all the elements of the MILP formulation when the

number of improvable arcs arrives close to zero. This reduction decreases with the number of arcs that can be improved. When 20% of the arcs can be improved, the preprocessing removes on average 80% and at least 70% of the model's variables, on average 60% and at least 45% of the model's constraints, and on average 75% and at least 65% of the model's non-zero entries. Even when 100% of the arcs can be improved, the preprocessing reduces on average by 35% the model's size. Since the gap between the 25th and 75th percentiles does not exceed 17% , the reduction seems to be robust. These results appear to be consistent with those of Table 1. Indeed, in the case of Aix, in which about 15% of the arcs can be improved, our preprocessing reduces the number of variables by 70% and the number of constraints by 65% .

### 5.3 Quality of the solutions

In this subsection, our goal is to compare the quality of solutions returned by the algorithms described in Section 4 to optimal solutions returned by the MILP solver.

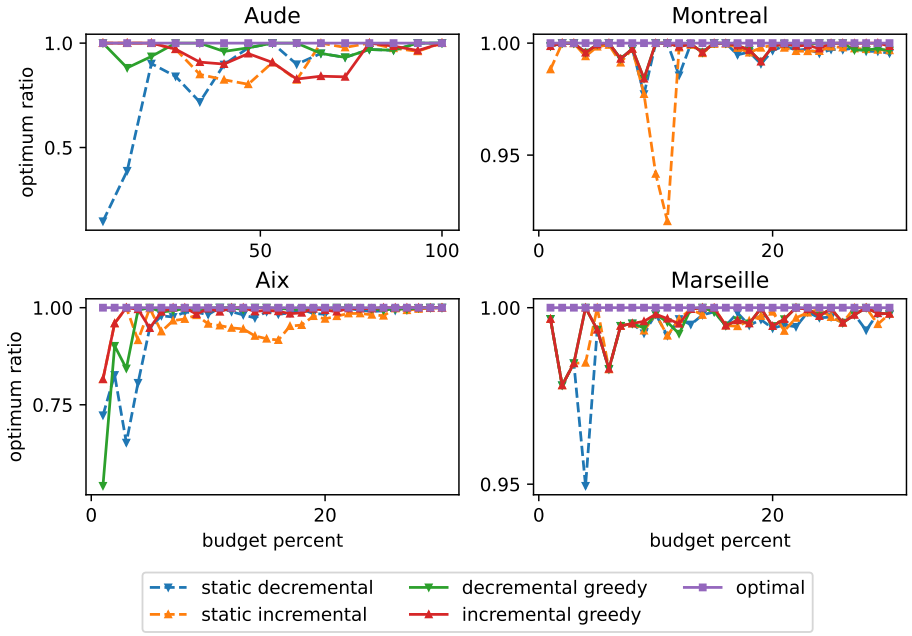



Figure 8: Percentage gain in ECA achieved by the solutions of the different algorithms compared to the optimal solution for different budget values.

For each case study and each of the four algorithms, there is at least one budget value for which the quality of the solution is significantly lower than the quality of the optimal solution, the greatest departures being observed at lower budget values (Figure 8, Table 3). Greedy versions of incremental and decremental algorithms

| | IL | | DL | | IG | | DG | |
|---|---|---|---|---|---|---|---|---|
| | min. | avg. | min. | avg. | min. | avg. | min. | avg. |
| Aude | 80.3 % | 94 % | 14.7 % | 83.9 % | 82.8 % | 94 % | 88.1 % | 97.2 % |
| Montreal | 92 % | 99.2 % | 97.7 % | 99.6 % | 98.4 % | 99.8 % | 98.4 % | 99.8 % |
| Aix | 81.6 % | 96.2 % | 65.2 % | 95.6 % | 81.6 % | 98.7 % | 54.1 % | 97.4 % |
| Marseille | 97.8 % | 99.5 % | 95 % | 99.3 % | 97.8 % | 99.6 % | 97.8 % | 99.6 % |

Table 3: Minimum and average optimilaty ratio for each algorithm and case study.

perform on average better than their static counterpart (Table 3). The minimum and average optimality ratio in the Aude and Aix cases is lower than in the other cases, for all algorithms (Table 3). This can be explained by the fact that the improvements seem to have a stronger impact on the distances between patches in the case of Aude and Aix. Static and greedy algorithms are generally quite close to the optimal solution (5% lower on average). However, all algorithms, whether static or greedy, incremental or decremental, provide poor quality solutions for some budget values (Table 3).

# 6 Conclusion

This article introduces a new MILP formulation for BC-ECA-OPT and shows that this formulation allows us to optimally solve instances having up to 150 habitat patches while previous formulations, such as those described in [29] are limited to 30 patches. The preprocessing step reduces significantly the size of the graphs on which a generalized flow has to be computed, thus enabling us to scale up to even larger instances. We showed that this preprocessing step allows us to greatly reduce the MILP formulation size and that its benefits increase when the proportion of arcs whose lengths can change is decreased. This allows us to tackle instances up to 300 habitat patches.

The optimum solutions obtained experimentally are compared to the ones returned by several greedy algorithms. Interestingly, we found that greedy algorithms work well in practice despite the arbitrarily bad cases we identified. Therefore, greedy algorithms remain a reasonable choice when the size of the instances is too large to be solved optimally by a MILP solver. Our next objectives will be to experiment with our approach on other practical instances of the problem arising from different ecological contexts.

On the theoretical side, we will investigate the problem from the point of view of an approximation algorithm, by addressing, e.g., the question of whether it is possible to find reasonable assumptions under which greedy algorithms are guaranteed to return a solution whose ECA value is at least a constant fraction of the optimal

ECA. If these assumptions are fulfilled by the real instances that we considered, this would explain our experimental observations. Moreover, since a polynomial time approximation scheme has been given in the case of trees [28], it might also be interesting to know for which larger classes of graphs there are constant factor approximation algorithms. Another interesting question is to determine whether good solutions could be obtained by geographically decomposing the problem, independently solving a subproblem for each region and then reassembling the solutions. In this case, a notion of fairness could help to allocate the budget among the regions so that each region can enhance its own *internal connectivity* keeping a part of the budget to enhance the connectivity between regions. Since the ECA is based on equivalence between a landscape and a patch, such a multilevel optimization approach looks promising.

## 7 Acknowledgments

We thank the reviewers and the editor for their careful reading and their many helpful comments and suggestions. The research on this paper was supported by Région Sud Provence-Alpes-Côte d'Azur, Natural Solutions and the ANR project DISTANCIA (ANR-17-CE40-0015). The Aix case study belongs to the Baum program (Biodiversity Urban Development Morphology) supported by the PUCA, the OFB and the DGALN. For stimulating exchanges on the case studies, we also thank: Patrick Bayle, Simon Blanchet, Andrew Gonzalez, Maria Dumitru, Jérôme Prunier, Bronwyn Rayfield, Benoit Romeyer and Keoni Saint-Pé.

## References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network flows: Theory, algorithms, and applications*, Prentice-Hall, Inc., 1993.

[2] C.H. Albert, B. Rayfield, M. Dumitru, and A. Gonzalez, *Applying network theory to prioritize multispecies habitat networks that are robust to climate and land-use change*, Conservation Biol. **31** (2017), 1383–1396.

[3] M. Awade, D. Boscolo, and J.P. Metzger, *Using binary and probabilistic habitat availability indices derived from graph theory to model bird occurrence in fragmented forests*, Landscape Ecology **27** (2012), 185–198.

[4] A. Billionnet, *Mathematical optimization ideas for biodiversity conservation*, Eur. J. Oper. Res. **231** (2013), 514–534.

[5] A. Billionnet, *Designing protected area networks*, EDP Sciences, Paris, France, 2021.

[6] E.S. Brondizio, J. Settele, S. Díaz, and H.T. Ngo, *Global assessment report on biodiversity and ecosystem services of the Intergovernmental Science- Policy Platform on Biodiversity and Ecosystem Services*, IPBES, Bonn, Germany, 2019.

[7] D. Catanzaro, M. Labbé, and M. Salazar-Neumann, *Reduction approaches for robust shortest path problems*, Comput. OR **38** (2011), 1610–1619.

[8] C. Demetrescu and G.F. Italiano, *A new approach to dynamic all pairs shortest paths*, J. ACM (JACM) **51** (2004), 968–992.

[9] B. Dezs, A. Jüttner, and P. Kovács, *Lemon - an open source c++ graph template library*, Electron. Notes Theor. Comput. Sci. **264** (2011), 23–45.

[10] L. Gurobi Optimization, 2022. *Gurobi optimizer reference manual.*

[11] J.O. Hanson, R. Schuster, M. Strimas-Mackey, and J.R. Bennett, *Optimality in prioritizing conservation projects*, Methods Ecology Evolution **10** (2019), 1655–1663.

[12] J.A. Jaeger, T. Soukup, C. Schwick, L.F. Madriñán, and F. Kienast, *Landscape fragmentation in Europe*, Publications office of the European Environmental Agency, Luxembourg, 2011.

[13] G.P. McCormick, *Computability of global solutions to factorable nonconvex programs: Part I - convex underestimating problems*, Math. Programing **10** (1976), 147–175.

[14] B.H. McRae, B.G. Dickson, T.H. Keitt, and V.B. Shah, *Using circuit theory to model connectivity in ecology, evolution, and conservation*, Ecology **89** (2008), 2712–2724.

[15] A. Moilanen, A.M. Franco, R.I. Early, R. Fox, B. Wintle, and C.D. Thomas, *Prioritizing multiple-use landscapes for conservation: Methods for large multi-species planning problems*, Proc. Royal Soc. B: Biological Sci. **272** (2005), 1885–1891.

[16] L. Pascual-Hortal and S. Saura, *Comparison and development of new graph-based landscape connectivity indices: Towards the priorization of habitat patches and corridors for conservation*, Landscape Ecology **21** (2006), 959–967.

[17] J. Pereira, S. Saura, and F. Jordán, *Single-node vs. multi-node centrality in landscape graph analysis: Key habitat patches and their protection for 20 bird species in ne spain*, Methods Ecology Evolution **8** (2017), 1458–1467.

[18] M. Pereira, P. Segurado, and N. Neves, *Using spatial network structure in landscape management and planning: A case study with pond turtles*, Landscape Urban Planning **100** (2011), 67–76.

[19] C. Pheatt, *Intel® threading building blocks*, J. Comput. Sci. Colleges **23** (2008), 298–298.

[20] L. Rubio, O. Bodin, L. Brotons, and S. Saura, *Connectivity conservation priorities for individual patches evaluated in the present landscape: How durable and effective are they in the long term?*, Ecography **38** (2015), 782–791.

[21] D.A. Rudnick, S.J. Ryan, P. Beier, S.A. Cushman, F. Dieffenbach, C.W. Epps, L.R. Gerber, J.N. Hartter, J.S. Jenness, J.A. Kintsch, A.M. Merenlender, R.M. Perkl, D.V. Preziosi, and S.C. Trombulak, *The role of landscape connectivity in planning and implementing conservation and restoration priorities*, Issues Ecology **16** (2012), 1–23.

[22] K. Saint-Pé, *In situ quantification of brown trout movements*, Ph.D. thesis, Université Paul Sabatier-Toulouse III, 2019.

[23] S. Saura, C. Estreguil, C. Mouton, and M. Rodríguez-Freire, *Network analysis to assess landscape connectivity trends: Application to european forests (1990-2000)*, Ecological Indicators **11** (2011), 407–416.

[24] S. Saura and L. Pascual-Hortal, *A new habitat availability index to integrate connectivity in landscape conservation planning: Comparison with existing indices and application to a case study*, Landscape Urban Planning **83** (2007), 91–103.

[25] S. Saura and J. Torne, *Conefor sensinode 2.2: A software package for quantifying the importance of habitat patches for landscape connectivity*, Environmental Modelling Software **24** (2009), 135–139.

[26] P.D. Taylor, L. Fahrig, K. Henein, and G. Merriam, *Connectivity is a vital element of landscape structure*, Oikos **68** (1993), 571–573.

[27] D. Urban and T. Keitt, *Landscape connectivity: A graph-theoretic perspective*, Ecology **82** (2001), 1205–1218.

[28] X. Wu, D. Sheldon, and S. Zilberstein, *Stochastic network design in bidirected trees*, Proceedings of the Twenty-Eighth Neural Information Processing Systems Conference, 2014, pp. 882–890.

[29] Y. Xue, X. Wu, D. Morin, B. Dilkina, A. Fuller, J.A. Royle, and C.P. Gomes, *Dynamic optimization of landscape connectivity embedding spatial-capture-recapture information*, 31st AAAI Conference on Artificial Intelligence, Vol. 31, 2017, pp. 4552–4558.