Constraints (2018) 23:335–354
https://doi.org/10.1007/s10601-018-9288-3

Intruder alert! Optimization models for solving the mobile robot graph-clear problem

Michael Morin[1,2] · Margarita P. Castro[2] · Kyle E. C. Booth[2] · Tony T. Tran[2] · Chang Liu[2] · J. Christopher Beck[2]

Published online: 21 May 2018

Abstract We develop optimization approaches to the graph-clear problem, a pursuit-evasion problem where mobile robots must clear a facility of intruders. The objective is to minimize the number of robots required. We contribute new formal results on progressive and contiguous assumptions and their impact on algorithm completeness. We present mixed-integer linear programming and constraint programming models, as well as new heuristic variants for the problem, comparing them to previously proposed heuristics. Our empirical work indicates that our heuristic variants improve on those from the literature, that constraint programming finds better solutions than the heuristics in run-times reasonable for the application, and that mixed-integer linear programming is superior for proving optimality. Given their performance and the appeal of the model-and-solve framework, we

This article belongs to the Topical Collection: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*
Guest Editor: Willem-Jan van Hoeve

† Margarita P. Castro and Kyle E.C. Booth equally contributing authors.

✉ Michael Morin
michael.morin@osd.ulaval.ca; mmorin@mie.utoronto.ca

Margarita P. Castro
mpcastro@mie.utoronto.ca

Kyle E. C. Booth
kbooth@mie.utoronto.ca

Tony T. Tran
tran@mie.utoronto.ca

Chang Liu
cliu@mie.utoronto.ca

J. Christopher Beck
jcb@mie.utoronto.ca

[1] Department of Operations and Decision Support Systems, Université Laval, Québec, QC, Canada

[2] Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON, Canada

conclude that the proposed optimization methods are currently the most suitable for the graph-clear problem.

Keywords Pursuit-evasion · Graph-clear problem · Constraint programming · Mixed-integer linear programming · Optimization · Mobile robotics

1 Introduction

The application of autonomous multi-robot systems (MRS) to real-world problems has recently received significant attention in academia and industry. One such application area is intelligent security systems, where the use of autonomous MRS has been proposed to provide facility security and monitoring [18, 23]. In addition to the development of physical robots, there has been simultaneous investigation into algorithms for *multi-robot surveillance* [19], concerning how an intelligent security system can be used to efficiently search complex environments for intruders. A classic problem in this area, the *pursuit-evasion problem* [20] has intruders actively trying to evade being detected by the searchers. Since this initial work, there have been numerous variants introduced, adding characteristics to further refine the problem definition to accurately represent real-world restrictions associated with mobile MRS.

We investigate the *graph-clear problem* (GCP), a previously proposed pursuit-evasion variant which models the problem of removing multiple intruders from an environment as an $\mathcal{NP}$-hard graph theoretic problem [11]. The goal is to find a schedule that minimizes the total number of robots needed to "clear" possible intruders from a given facility, represented as a graph. The graph is a discretization of the real-world environment, representing a floor plan, where the clearing operations occur [12]; a graph and corresponding environment for an example instance is illustrated in Fig. 1. To clear the graph, the team of robots execute *sweep* and *block* actions on nodes and edges, respectively. Sweep actions remove intruders from *contaminated* nodes (regions potentially containing intruders) and block actions prevent intruders from traveling between nodes. The execution of these actions may require multiple robots, represented on the graph with node and edge weights. During the sweeping of a node, all of the edges that connect it to other nodes must be blocked in order to prevent *recontamination* by potential intruders re-entering a node. A solution to the GCP is a schedule of sweep and block actions that detects all potential intruders in the facility (i.e., "clears" the facility) while minimizing the number of robots required. A solution that prevents recontamination and, therefore, removes the need to sweep a node more than once, is

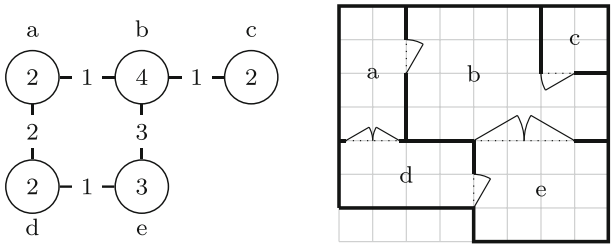


Fig. 1 A GCP instance (graph and floor plan adapted from [14]), its weighted graph (left) and corresponding facility floor plan (right); node and edge weights indicate the number of robots required to sweep and block those elements, respectively

called *progressive*. Solutions such that cleared vertices at each time step form a connected subgraph are termed *contiguous*. Existing approaches for the GCP are custom heuristics that produce a single solution, the quality of which is only known *w.r.t.* static, separately derived bounds.

In this paper we: (i) provide new formal results on the impact of progressive and contiguous assumptions on algorithm completeness, (ii) develop and empirically evaluate *mixed-integer linear programming* (MILP) and *constraint programming* (CP) models for the GCP, and (iii) propose novel variations of existing heuristics. The MILP and CP techniques employ branch-and-bound search and benefit from the model-and-solve paradigm. Once the problem has been expressed in either formalism, high-powered solvers are used to find a series of improving solutions while enhancing bounds throughout the search. Alternative problem variations can be represented with adjustments to the model and do not require newly customized algorithms.

The remainder of the paper is structured as follows: Section 2 formally defines the GCP while Section 3 identifies related work. Section 4 presents new formal results related to progressive and contiguous solution assumptions and Section 5 presents both exact and heuristic techniques for solving the GCP. Section 6 presents an empirical evaluation of the methods and Section 7 provides concluding remarks and directions for future research.

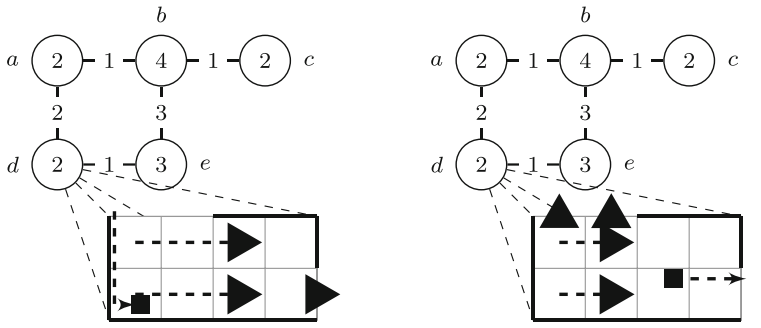
2 Problem definition

A GCP instance is defined as an undirected weighted graph $G = (V, E, a, b)$ where $a : V \rightarrow \mathbb{N}^+$ and $b : E \rightarrow \mathbb{N}^+$ are weights over the nodes, $v \in V$, and edges, $e \in E$, respectively. Nodes and edges can be either clear or contaminated. A node or edge is clear when it is known with certainty that it does not contain an intruder, otherwise it is considered contaminated. The robot team can sweep nodes and block edges to clear the environment. Weight a_v represents the number of robots required to sweep a node $v \in V$, while weight b_e indicates the number of robots needed to block an edge $e \in E$. As before, a solution to the GCP is a schedule of sweep and block actions that detects all intruders, and thus clears the environment.

As originally presented [11], clearing a node requires the robot team to perform a sweep action on that node while blocking all incident edges. This requirement is driven by the sweep patterns that can be implemented by real-world robots using physical sensors, often with limited (finite) detection capabilities. Figure 2 illustrates a sweep pattern of a region swept by two robots. Blocking all incident edges during a sweep action prevents: (i) the node currently being swept from becoming immediately recontaminated from another contaminated region (Fig. 2a) and (ii) an intruder within the region being swept from escaping into a previously cleared region (Fig. 2b).

As before, progressive solutions do not allow recontamination (and, therefore, do not require a node to ever be re-swept) and the cleared nodes in contiguous solutions form connected subgraphs at each step.

Example We illustrate a GCP instance and solution in Fig. 3, where the environment is cleared in five steps using a team size of 10 robots. A total of five robots are needed to perform step 1 (two for sweeping node a and three for blocking edges connected to node a, as in Fig. 3a). Step 2 requires six robots: two to sweep node d, two to prevent intruders from escaping from node d to node a, one to prevent intruders from entering node d from node e and one to prevent intruders from entering node a from node b (Fig. 3b). The cost of



(a) Intruder recontaminating the region during sweeping.

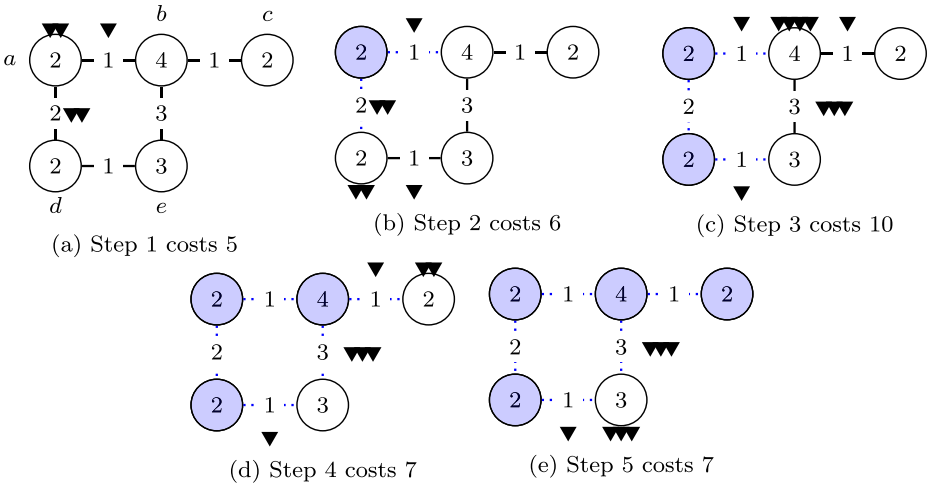
(b) Intruder escaping the region during sweeping.

Fig. 2 Sweep patterns: Blocking incident edges during sweeping of node d to avoid immediate recontamination. Intruders represented by black squares and robots by triangles. Movement patterns of intruder and sweeping robots denoted with dotted lines

the solution, 10 robots, corresponds to the number of robots at the third step: the maximum number of robots used over all steps. This solution, expressed as $\Sigma = (a, d, b, c, e)$, is progressive since no recontamination occurs, and contiguous as cleared nodes form connected subgraphs at all steps.

2.1 Formal definitions and notation

To compactly express our formal results and optimization models, we introduce additional notation. Let τ represent the set of discretized steps. For each step $t \in \tau$, let σ_t represent the node swept, B_t the set of edges blocked, C_t the set of clear nodes, and D_t the set of clear edges. A solution to a GCP instance is then a sequence of tuples $\Pi_\tau = \langle \pi_0, \pi_1, \ldots, \pi_{|\tau|} \rangle$ where $\pi_t = (\sigma_t, B_t, C_t, D_t)$ for all $t \in \tau$. Formally, a node or edge is clear at step $t \in \tau$,



(a) Step 1 costs 5

(b) Step 2 costs 6

(c) Step 3 costs 10

(d) Step 4 costs 7

(e) Step 5 costs 7

Fig. 3 A GCP problem instance and five-step solution of cost 10 (incurred in step 3). Cleared nodes are blue, cleared edges are dotted, and each individual robot is represented by a triangle. (Color figure online)

iff it belongs to the set C_t or D_t, respectively. Otherwise, it is contaminated. At $t = 0$, $\pi_0 = (\emptyset, \emptyset, \emptyset, \emptyset)$. For a given step $t \in \tau$, the following actions can be performed: (i) a node $v \in V$ is swept iff $\sigma_t = v$; (ii) an edge $e \in E$ is blocked at iff $e \in B_t$.

Let $path(x, y)$ be a finite sequence of incident nodes and edges without repetition starting from $x \in V \cup E$ and ending at $y \in V \cup E$.[1] An *open path* (also called recontamination path), $opath(x, y)$, is a path from $x \in V \cup E$ to $y \in V \cup E$ that does not contain any blocked edges. The state of the edges and nodes evolve according to the following formal definitions.

Clearing an edge An edge $e \in E$ is clear at step $t \in \tau$ (i.e., $e \in D_t$) iff either (i) it is blocked at t or (ii) it is clear at $t - 1$ and all open paths at step t containing e do not contain contaminated nodes or edges. That is,

$$e \in D_t \Leftrightarrow e \in B_t \\ \vee (e \in D_{t-1} \wedge \forall opath(v', e) : v' \in C_t \ \wedge \ \forall opath(e', e) : e' \in D_t) .$$

Clearing a node A node $v \in V$ is clear at step $t \in \tau$ (i.e., $v \in C_t$) iff either (i) it is clear at $t - 1$, and all open paths at step t containing v do not contain contaminated nodes or edges or (ii) it is swept at step t and all its adjacent edges, denoted by $\delta(v)$, are blocked. That is,

$$v \in C_t \Leftrightarrow (v \in C_{t-1} \ \wedge \ \forall opath(v', v) : v' \in C_t \ \wedge \ \forall opath(e', v) : e' \in D_t) \\ \vee (\sigma_t = v \ \wedge \ \forall e \in \delta(v) : e \in B_t) .$$

Solution cost, feasibility, and optimality A solution Π_t is feasible iff there exists a $t \in \tau$ such that $C_t = V$ and $D_t = E$ (i.e., all nodes and edges are clear at some step t). The cost of the solution is defined as $z(\Pi_\tau) = \max_{t \in \tau} \{z(\pi_t)\}$, where $z(\pi_t) = a_{\sigma_t} + \sum_{e \in B_t} b_e$ is the cost at step $t \in \tau$. An optimal solution Π_τ minimizes $z(\Pi_\tau)$ while remaining feasible.

2.2 Assumptions

We make the following assumptions in this paper:

1. Solutions are progressive (i.e., recontamination is not permitted).
2. A single node, $v \in V$, is swept at each step, $t \in \tau$.
3. No extra edges are blocked beyond those to assure no recontamination.

Enforcing no more than a single sweep at a time (Assumption 2) preserves algorithm completeness [14]. It can be seen that single node sweeping and minimal edge blocking (Assumption 3) preserve completeness when assuming progressiveness. Somewhat surprisingly, the impact of the first assumption on completeness remains an open question in the general case, perhaps due to the unusual objective function as compared with more standard graph problems.

Under these assumptions, it is sufficient to define a solution in terms of the node swept at each step. As such, the graph will be cleared in exactly $|V|$ steps. The set of blocked edges at step $t \in \tau$ follows directly from the node swept at that step. Following Kolling and Carpin [14], we define $\delta(X)$, with $X \subset V$, as the subset of edges with a node in X and a node in $V \setminus X$. When sweeping a single node $v \in V$ at step t, the set of edges preventing

[1] The definition of path used in the graph-clear literature is a generalization of the classical definition in that it is permitted to start or end on an edge.

recontamination is $\delta(C_{t-1}) \cup \delta(v)$ (i.e., edges incident to v, $\delta(v)$, and all edges between C_{t-1} and $V \setminus C_{t-1}$).

It follows that, under these conditions, a solution can be expressed as a sequence of nodes $\Sigma = (\sigma_1, \ldots, \sigma_n)$ where $n = |V|$. The cost in each step $t \in \{1, \ldots, n\}$, $z(\sigma_t)$, and the cost of the solution, $z(\Sigma)$, are given by:

$$z(\sigma_t) = a_{\sigma_t} + \sum_{e \in \delta(C_{t-1}) \cup \delta(\sigma_t)} b_e \ , \qquad z(\Sigma) = \max_{t \in \{1,..,n\}} \{z(\sigma_t)\} \ . \tag{1}$$

3 Related work

As a multi-agent search for adversarial targets in a discrete, finite environment [4], the GCP is a pursuit-evasion problem variant. In the GCP, the search, or pursuit, is performed under a worst-case assumption on target (intruder) motion[2] and a finite range, perfect detection assumption on agent (pursuer) detection capabilities. Given the large body of pursuit-evasion literature, we limit our review to the works that directly apply to the GCP. The study of these problems is important for the development of intelligent security systems, which have been proposed to provide facility security and monitoring [18, 23] using autonomous multi-robot teams.

The GCP was first investigated in the context of a graph theoretic problem dubbed *weighted graph clearing* [10]. The problem permits node recontamination, simultaneous sweeping of multiple nodes, and non-contiguous solutions. In this previous work, an algorithm is presented that reduces the problem graph into a tree and implements a heuristic tree-clearing strategy with a single node swept at each step [10]. The reduction is achieved by calculating a minimum spanning tree (MST) and then blocking all other edges such that only the MST remains (carrying the cost of this blocking as a constant). The work was then extended with the official coining of the term "graph-clear" and an initial claim that the GCP, for the general case of graphs, is an $\mathcal{NP}$-hard problem. The authors also present a contiguous version of the heuristic clearing algorithm for trees [11] with a single node swept at each step, inspired by the graph searching literature [1].

More recent work demonstrated that a hybrid heuristic, using a combination of contiguous and non-contiguous clearing strategies on greedily determined partitions, outperforms both of the standalone heuristic methods [13]. Full details were included formally demonstrating that, for the general case of graphs, the GCP is $\mathcal{NP}$-hard, and an $O(n^2)$-time algorithm for trees with n nodes was developed under the contiguous solution assumption [14]. Supplementary formal work showed that, for any graph, multi-node sweeping provides no advantage, and when the graph is a tree and contiguity is enforced, the progressive assumption does not decrease solution quality, a result that is also conjectured to hold for general graphs [14].

As noted in these previous works, the definition of the GCP has strong roots in edge and node search problems [5, 9, 20]. These problems involve finding a pursuer strategy that will guarantee finding the intruders under a perfect detection assumption. The main difference between edge and node search and the GCP resides in the actions available to the pursuers. In the GCP, pursuers can perform sweep and block actions, while node and edge search involves sliding along or barring both sides of an edge. Sweeping and blocking are

[2] The distinction between single vs. multiple targets makes no difference in our context.

more practical assumptions for mobile robotics [14], making the GCP abstraction closer to real-world problems.

Optimization-based techniques, namely MILP and CP, have been previously proposed for problems involving multi-robot teams [3, 16]. With regards to the GCP, MILP has been recently applied to the problem of minimizing the execution time of a given graph-clearing solution [22] without increasing the robot fleet size, a different problem than that studied in this work.[3] To our knowledge, MILP and CP has not been previously proposed to find a graph-clearing schedule.

Model checking was also proposed for solving pursuit-evasion problems in a paper that features a graph-clear model and a weighted edge-searching model [21]. The approach involves running the model checker multiple times, increasing a fixed upper bound on the objective value each time. The solution is optimal when the model checker proves a bound to be feasible. The performance of the approach is demonstrated on a single instance with five nodes and five edges, which is solved in less than a second. The paper presents no empirical evidence of how the approach scales to larger graphs.

4 Formal results

In this section we present new formal results for two of the most common GCP assumptions: progressiveness and contiguity.

4.1 Progressive solutions

We provide new results to support our first assumption regarding solution progressiveness. To achieve this, we show that two specific recontamination types, if permitted to occur, would not improve the quality of a solution.

Consider a GCP instance $G = (V, E, a, b)$ and a feasible solution Π_τ. For a given step $t \in \tau$, consider the subgraph induced by C_t, and let $C_t(v)$ be the connected component of the subgraph that includes node $v \in C_t$. We say that Π_τ allows recontamination at step t if it has a recontamination edge (i.e., there exists an $e = \{u, v\} \in \delta(C_{t-1}) \cup \delta(\sigma_t)$ with $v \in C_{t-1}$ and $e \notin B_t$).

We define three types of recontamination that together define all possible cases. The recontamination is *complete* if, for each recontamination edge $e = \{u, v\}$, all nodes in $C_{t-1}(v)$ are recontaminated. The recontamination is *partial* if a subset of $C_{t-1}(v)$ is contaminated (i.e., a subset of edges in $C_{t-1}(v)$ is blocked). Lastly, the recontamination is *single-node* if only one node is recontaminated at step t.

Proposition 1 *Consider a GCP instance $G = (V, E, a, b)$ and a feasible solution $\Pi_\tau = \langle \pi_1, \ldots, \pi_{|\tau|} \rangle$ with $\pi_t = (\sigma_t, B_t, C_t, D_t)$. Let all recontaminations be complete. Then, there exists a progressive feasible solution $\Sigma = (\sigma'_1, \ldots, \sigma'_n)$ such that $z(\Sigma) \leq z(\Pi_\tau)$.*

Proof First, consider that Π_τ has exactly one complete recontamination and it occurs at step t. Consider R_t as the set of nodes recontaminated at step t (i.e., $R_t \subseteq C_{t-1}$ and $R_t \subseteq V \backslash C_t$). Since Π_τ is feasible, all nodes in R_t are swept again after step t. Consider $\Sigma = (\sigma'_1, \ldots, \sigma'_n)$ that follows the sweeping order of Π_τ but omits clearing the nodes in R_t before t, which

[3] In this paper we consider only the GCP that minimizes the number of robots used.

is a feasible solution. Since the recontamination is complete, nodes in R_t are not connected to the nodes in $C_{t-1} \backslash R_t$. Hence, the cost of clearing nodes inside $C_{t-1} \backslash R_t$ in solution Σ is less than or equal to their clearing cost in Π_τ. By construction, all nodes not in $C_{t-1} \backslash R_t$ have the same cost in both solutions, and so $z(\Sigma) \leq z(\Pi_\tau)$.

We extend the proof to consider any number of recontaminations. For each recontamination step t, construct a new strategy that omits step t as describe above. Since the solution cost is maintained or reduced every time we remove a recontamination step, the resulting solution Σ satisfies $z(\Sigma) \leq z(\Pi_\tau)$. □

Proposition 2 *Consider a GCP instance $G = (V, E, a, b)$ and a feasible solution $\Pi_\tau = \langle \pi_1, \ldots, \pi_{|\tau|} \rangle$ with $\pi_t = (\sigma_t, B_t, C_t, D_t)$. Let all recontaminations be single-node. Then, there exists a progressive feasible solution $\Sigma = (\sigma'_1, \ldots, \sigma'_n)$ such that $z(\Sigma) \leq z(\Pi_\tau)$.*

Proof Without loss of generality, consider that Π_τ has exactly one single-node recontamination at step t (i.e., $|\tau| = n+1$). As in Proposition 1, the proof can be extended to consider multiple single-node recontamination.

Let v be the node recontaminated at step t, $t' < t$ the step when v was first swept, and $t'' > t$ the step where v is swept again. Consider $\delta^b(v)$ as the set of edges incident to v that were blocked at step t, and $\delta^r(v)$ be the set of recontamination edges (i.e., $\delta(v) = \delta^b(v) \cup \delta^r(v)$).

Consider the case where $\sum_{e \in \delta^b(v)} b_e \geq \sum_{e \in \delta^r(v)} b_e$ and $\Sigma = (\sigma'_1, \ldots, \sigma'_n)$ is a progressive solution which follows the clearing order of Π_τ and avoids recontamination at step t (i.e., $\sigma'_k = \sigma_k$ for $k < t''$ and $\sigma'_k = \sigma_{k+1}$ for $t'' \leq k \leq n$). By construction we have that $z(\sigma'_k) = z(\pi_k)$ for all $k < t$ and $z(\sigma'_k) = z(\pi_{k+1})$ for all $t'' \leq k \leq n$. Due to the edge cost assumption we get $z(\sigma'_k) \leq z(\pi_k)$ for all $t \leq k < t''$. Therefore, $z(\Sigma) \leq z(\Pi_\tau)$.

Now consider $\sum_{e \in \delta^b(v)} b_e < \sum_{e \in \delta^r(v)} b_e$ and a progressive solution $\Sigma = (\sigma'_1, \ldots, \sigma'_n)$ with $\sigma'_k = \sigma_k$ for $k < t'$ and $\sigma'_k = \sigma_{k+1}$ for $t' \leq k \leq n$ (i.e., avoid sweeping v at step t'). By construction, $z(\sigma'_k) = z(\pi_k)$ for all $k < t'$ and $z(\sigma'_k) = z(\pi_{k+1})$ for all $k \geq t$. Since the edge cost assumption guarantees that $z(\sigma'_k) \leq z(\pi_k)$ for $t' \leq k \leq t$, we have that $z(\Sigma) \leq z(\Pi_\tau)$. □

Propositions 1 and 2 show that neither complete nor single-node recontamination steps can improve solution quality. It remains to prove the case of a *partial* recontamination. We can extend Proposition 2 to consider partial recontamination iff the recontaminated nodes are swept in a sequence both before and after the recontamination occurs. With this consideration, we can then treat the recontaminated nodes as single-node cases. While the general case remains open, we conjecture that the progressive assumption does not sacrifice the completeness of the algorithm.

4.2 Contiguous solutions

We now restrict our focus to progressive and contiguous solutions, which have received particular attention in the literature [15]. Given a graph $G = (V, E)$ and a feasible progressive solution $\Sigma = (\sigma_1, \ldots, \sigma_n)$, we say that Σ is *contiguous* iff $\delta(\sigma_t) \cap \delta(\{\sigma_1, \ldots, \sigma_{t-1}\}) \neq \emptyset$ for all $1 < t \leq n$ (i.e., the set of clear nodes at each step forms a connected subgraph).

To our knowledge, no previous work has shown the impact of enforcing contiguity. We provide a proof through a counter-example that enforcing contiguity may remove all optimal solutions. Consider the graph shown in Fig. 4. The optimal contiguous solution is $\Sigma^c =$

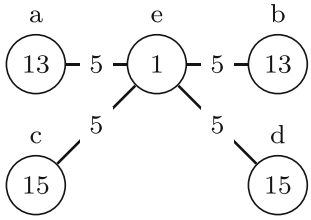


Fig. 4 A graph where enforcing contiguity prunes all optimal solutions

(a, e, b, c, d) with cost $z(\Sigma^c) = 28$. However, the optimal solution is $\Sigma = (c, a, e, b, d)$, which is not contiguous, and has a cost $z(\Sigma) = 23$.

Complete graphs and paths are structures where enforcing contiguity preserves optimality. Also, any graph with a node $v \in V$ satisfying $a_v + \sum_{e\in\delta(v)} b_e \geq a_{v'} + \sum_{e\in E} b_e$, for all $v' \in V\backslash\{v\}$ will not lose optimal solutions with contiguity constraints. Since such graphs have a node v that dominates all the rest *w.r.t.* cost, any solution that initially clears v is optimal.

5 Exact and heuristic methods

This section presents exact and heuristic methods to solve the GCP. We propose one MILP and two CP formulations of the problem. In addition, we provide novel variations for a set of heuristics in the literature [11].

We use the following notation in our MILP and CP models. Variable sets are in uppercase italic font and the indices are in square brackets (e.g., the i^{th} variable of *VAR* is *VAR*[i]). We use $H = \{1, \ldots, n\}$ as the set of steps.

The objective function in each model is given by variable Z, which has integer or continuous domain depending on the solver. Variable Z is bounded by $\underline{Z} = 1$ and $\overline{Z} = \max_{v\in V}\{a_v\} + \sum_{e\in E} b_e$ which are, respectively, the minimum and maximum number of robots needed to clear any graph. A tighter upper bound can be found solving a weighted maximum cut problem, which is $\mathcal{NP}$-hard in the general case [7]. Therefore, we refrained from further tightening the domain of Z.

5.1 Mixed-integer linear programming model

Our MILP model uses variable $X[i, t]$ to indicate whether or not node $i \in V$ is cleared at step $t \in H$, and $Y[i, j, t]$ to indicate if edge $\{i, j\} \in E$ is blocked at step $t \in H$. Continuous variable Z represents the objective function.

$$\min\ Z \qquad \text{(MILP)}$$

$$\text{s.t. } Z \geq \sum_{i\in V} a_i X[i, 1] + \sum_{\{i,j\}\in E} b_{\{i,j\}} Y[i, j, 1] \qquad (2)$$

$$Z \geq \sum_{i\in V} a_i \left(X[i, t] - X[i, t-1]\right) + \sum_{\{i,j\}\in E} b_{\{i,j\}} Y[i, j, t] \qquad \forall t \in H\backslash\{1\} \qquad (3)$$

$$\sum_{i \in V} X[i,t] = t \qquad \forall t \in H \quad (4)$$

$$X[i,t] \leq X[i,t+1] \qquad \forall i \in V,\ \forall t \in H \backslash \{n\} \quad (5)$$

$$X[i,t] - X[j,t] \leq Y[i,j,t] \qquad \forall \{i,j\} \in E, t \in H \quad (6)$$

$$X[j,t] - X[i,t] \leq Y[i,j,t] \qquad \forall \{i,j\} \in E,\ t \in H \quad (7)$$

$$X[i,t] - X[i,t-1] \leq Y[i,j,t] \qquad \forall \{i,j\} \in E,\ t \in H \backslash \{1\} \quad (8)$$

$$X[i,t] \in \{0,1\} \qquad \forall i \in V,\ t \in H \quad (9)$$

$$Y[i,j,t] \in \{0,1\} \qquad \forall \{i,j\} \in E,\ t \in H \quad (10)$$

$$\underline{Z} \leq Z \leq \overline{Z}, Z \in \mathbb{R}^+ \quad (11)$$

Constraint (2) represents the minimum number of robots needed to clear the node in the first step. Constraint (3) provides a lower bound on the necessary robots to clear a node in a progressive strategy (i.e., it implies (1)). Constraint (4) enforces that t nodes are cleared by step t. Constraint (5) enforces no recontamination (i.e., a node stays cleared after being swept). Similarly, constraints (6) and (7) enforce no recontamination on edges. Constraint (8) models the blocking of all incident edges for a node swept at step t. Lastly, constraints (9) to (11) express the domains of the variables.

5.2 Constraint programming models

We develop two CP models for the GCP. The first model, CPS, represents the sequence of swept nodes as variables (i.e., the t^{th} variable represents the node swept at position t in the sequence). The second model, CPN, uses one variable per node on the graph indicating the step at which the node is swept.

5.2.1 Sequence-based CP model (CPS)

Consider $W[t] \in V$ to be a decision variable indicating the node swept at step t in the sequence. The objective value is given by Z with integer domain.

$$\min\ Z \quad \text{(CPS)}$$

$$\text{s.t. } \textsc{AllDifferent}(W) \quad (12)$$

$$Z \geq a_{W[t]} + \sum_{e \in \delta(W[t])} b_e + \sum_{e \in \delta(\{W[1],\ldots,W[t-1]\}) \backslash \delta(W[t])} b_e \qquad \forall t \in H \quad (13)$$

$$W[t] \in V \qquad \forall t \in H \quad (14)$$

$$\underline{Z} \leq Z \leq \overline{Z},\ Z \in \mathbb{Z}^+ \quad (15)$$

Constraint (12) enforces each node to be in the sequence exactly once using the AllDifferent global constraint [17]. Constraint (13) provides a lower bound on Z, related to the node sweeping and edge blocking costs at each step. This constraint makes extensive use of the Element constraint [24], permitting the indexing of arrays with decision variables, to attain the cost values. For a given step t, $a_{W[t]}$ represents the robots required to sweep node $W[t]$; we note the use of the Element global constraint as the array a is indexed by decision variable $W[t]$. The second term on the RHS of constraint (13) indicates the number of robots to block all edges incident to node $W[t]$. The term $\delta(W[t])$ again

uses the ELEMENT global constraint to identify the set of edges incident to the node swept at step t. Recall that $\delta(X)$, with $X \subset V$, is the subset of edges with a node in X and a node in $V \setminus X$. The third term on the RHS of constraint (13) indicates the number of additional robots required to prevent previously cleared nodes from being recontaminated during step t. This term is equal to zero for $t \in \{1, n\}$ as recontamination is not possible for these steps: for $t = 1$, no prior nodes have been cleared to be recontaminated and for $t = n$ there are no more intruders to perform the recontamination (the only remaining intruders are those in the node being swept at step n). Lastly, constraints (14) and (15) express the variable domains.

5.2.2 Node-based CP model (CPN)

Let $T[i] \in H$ be a variable representing the step at which node $i \in V$ is swept. Let variables $L[i, j] \in H$ and $U[i, j] \in H$ indicate the step at which an edge $\{i, j\} \in E$ is blocked and released, respectively. We denote $c_{\text{sweep}} = \max_{i \in V}\{a_i + \sum_{e \in \delta(i)} b_e\}$ as the maximum cost of sweeping any node while blocking its incident edges and $c_{\text{block}} = \sum_{e \in E} b_e$ as the cost of blocking all edges. Then, $S[t] \in \{1, \ldots, c_{\text{sweep}}\}$ and $B[t] \in \{1, \ldots, c_{\text{block}}\}$ are variables representing the sweeping and blocking cost of the solution at step $t \in H$, respectively. Also, let $I[i, j, t] \in \{0, 1\}$ indicate whether or not blocking edge $\{i, j\} \in E$ is necessary at $t \in H$. The objective value is given by Z with integer domain. Model CPN is shown below.

$$\min \ Z \tag{CPN}$$

$$\text{s.t. } Z = \max_{t \in H}\{S[t] + B[t]\} \tag{16}$$

$$T[i] = t \Rightarrow S[t] = a_i + \sum_{e \in \delta(i)} b_e \qquad \forall i \in V, t \in H \tag{17}$$

$$\text{ALLDIFFERENT}(T) \tag{18}$$

$$T[i] < T[j] \Rightarrow L[i, j] = T[i] \qquad \forall \{i, j\} \in E \tag{19}$$

$$T[i] < T[j] \Rightarrow U[i, j] = T[j] \qquad \forall \{i, j\} \in E \tag{20}$$

$$(L[i, j] \leq t \wedge U[i, j] \geq t) \wedge (T[i] \neq t \wedge T[j] \neq t) \Rightarrow I[i, j, t] = 1 \qquad \forall \{i, j\} \in E,\ t \in H \tag{21}$$

$$B[t] = \sum_{\{i,j\} \in E} b_{\{i,j\}} I[i, j, t] \qquad \forall t \in H \tag{22}$$

$$T[i],\ L[i, j],\ U[i, j] \in H \qquad \forall i \in V,\ \{i, j\} \in E \tag{23}$$

$$S[t] \in \{1, \ldots, c_{\text{sweep}}\} \qquad \forall t \in H \tag{24}$$

$$B[t] \in \{1, \ldots, c_{\text{block}}\} \qquad \forall t \in H \tag{25}$$

$$I[i, j, t] \in \{0, 1\} \qquad \forall i, j \in V,\ t \in H \tag{26}$$

$$\underline{Z} \leq Z \leq \overline{Z},\ Z \in \mathbb{Z}^+ \tag{27}$$

Constraint (16) fixes the value of Z to be the maximum number of agents needed. Constraint (17) ensures the correct sweeping cost at each step. Constraint (18) enforces all sweeping steps to be different. Constraint (19) ensures that the step at which an edge is blocked equals the earliest sweeping step of one of its adjacent nodes. Similarly, constraint (20) ensures that the step at which an edge is released equals the latest sweeping step of one of its adjacent nodes. Constraint (21) ensures that the indicator variable is well defined (i.e., an edge is blocked as soon as a block is enforced and the block stops as soon as it is released). The blocking cost at step t is encoded in constraint (22). Lastly, constraints (23) to (27) express the domains of the decision variables.

5.3 Enforcing contiguous solutions

For our MILP model, we enforce contiguous solutions using constraint (28). The resulting model, MILP-C, is given by (2)–(11) and (28).

$$\sum_{j \in V : \{i,j\} \in E} X[j, t-1] \geq X[i,t] - X[i, t-1] \qquad \forall i \in V, t \in H \backslash \{1\} \tag{28}$$

Constraint (29) enforces contiguous solutions for model CPS, while constraint (30) enforces it for model CPN. Both constraints can be modeled using multiple InRelation global constraints [2], also called Table constraints, to explicitly limit the allowed tuples of values for the variable pairs. Then, CPS-C is given by (12)-(15),(29), and CPN-C is given by (16)-(27), (30).

$$\bigvee_{k=1}^{t-1} \{W[k], W[t]\} \in E \qquad \forall t \in H \backslash \{1\} \tag{29}$$

$$T[i] > 0 \Leftrightarrow \bigvee_{j \in V : \{i,j\} \in E} (T[j] < T[i]) \qquad \forall i \in V \tag{30}$$

5.4 Tree-based heuristics for graph-clearing

A variety of heuristics for variants of GCP have been proposed in the literature. However, due to the diversity of the literature, there does not appear to be a clear "best" heuristic to compare against. As a consequence, we adopt the main heuristic form present in the literature and experiment with a number of components, both existing and novel. The former are representative of the current state of the art.

The existing heuristics for the GCP [11] are composed of three main steps:

1. Transform $G = (V, E)$ into a spanning tree (ST) $T = (V, E')$ by removing edges;
2. Find a solution on T; and
3. Construct a solution for G using the one found on T.

Finding the ST that results in the best solution for G (though there are no optimality guarantees) is an open question [15]. Furthermore, it is unknown which characteristics of a solution on T will lead to the best solution on G. We now describe the components selected from the literature to build our heuristics and identify the novel contributions as appropriate.

Transform G into a spanning tree T We experimented with both minimum and maximum STs. The use of a minimum spanning tree of G is prevalent in the literature [11, 15] while maximum STs do not seem to have been considered. Our intuition is that a maximum ST may provide a stronger relaxation of G as it removes the cheapest edges which are likely to have less impact on the solution cost of G.

Find a solution on T In order to find a solution on T, we used a node labeling heuristic from the literature [11]. The heuristic computes the *root-cost matrix* $\mathbf{R}$ defined over each pair of nodes $x, y \in V$, where $\mathbf{R}_{x,y}$ stores the cost of recursively clearing the subtree rooted in y when entering from x [11]. One can define different heuristics using $\mathbf{R}$. Given a root node v, sweeping the nodes recursively starting from the leaves leads to a non-contiguous solution in most cases, while sweeping the nodes from the root enforces contiguity [11].

In addition, one has to select a root node. Kolling and Carpin [11] picked the root in the center of the longest path in T. Our implementation iterates through all nodes, generates a heuristic solution with each one as the root, and keeps the best solution (i.e., we evaluate n solutions per instance). It can be seen that this choice can only improve the objective value *w.r.t.* picking the root in the middle of the longest path in T.

Construct the strategy for G Adapting a solution found for T to G is straightforward when assuming progressive solutions. The edges not in T, i.e., $E \setminus E'$, are called cycle edges. The first approach discussed in the literature is called static blocking [11] which involves blocking the cycle edges during the entire execution of the solution to T on G. We use the *dynamic blocking* approach that follows the node order of the solution to T, while blocking only the cycle edges necessary to prevent recontamination [11].

Implemented heuristics In summary, the implementation decisions are:

- test both maximum and minimum STs;
- run heuristic multiple times, choosing each node as the ST root;
- start sweeping at the root node when enforcing contiguity and start at the leaf nodes in the general case; and,
- use dynamic edge blocking.

From these implementation decisions, using the maximum spanning and looping over all nodes to find the best root are novel contributions while the last two implementation decisions are common to the graph-clear literature.

These choices lead to four heuristics, namely a minimum ST heuristic that guarantees contiguity (MinSTH-C), a maximum ST heuristic that guarantees contiguity (MaxSTH-C), a general minimum ST heuristic (MinSTH), and a general maximum ST heuristic (MaxSTH). Heuristics MaxSTH and MaxSTH-C both use the maximum ST of G, whereas MinSTH and MinSTH-C use the minimum ST of G. The heuristics suffixed by C are guaranteed to return a contiguous solution.

6 Experiments

Since the GCP is an off-line problem where decreasing the number of robots can substantially reduce the cost of an operation, we argue that allocating more time to find a solution is reasonable. In contrast to the heuristic approaches, our methodology is complete, subject to the assumptions made.

All models were run on an Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz using a single thread per run. We enforced a time limit of 1-hour on every run after which the solver provides its incumbent solution. The MILP models were solved using IBM ILOG CPLEX 12.7.1. The CP models were solved using IBM ILOG CPOptimizer 12.7.1. The generation of the models was done in C++ using IBM ILOG Concert Technology. We use non-default variable and value selection heuristics for the sweep variables in the CP models. Both CP models order variables based on the smallest domain heuristic. CPN instantiates the chosen variable using the smallest value heuristics while CPS instantiates the chosen variable using the impact-based heuristic of CPOptimizer. When instantiating the chosen variable, this heuristic selects the value leading to the smallest average reduction of the search space observed so far. The MILP model is solved using the default configuration except for preprocessing where we set CPLEX to use aggressive symmetry breaking. All parameters were

chosen using preliminary experiments on small problem instances. All heuristics were run in Python.[4]

We compared the performance of our models to the ones obtained by the heuristics in terms of objective value, both for the general case and when restricting to contiguous solutions. In addition, we present results when the optimization models are warm-started with the solution from a heuristic, i.e., the solution of the heuristic is used as a starting solution. We chose MaxSTH and MaxSTH-C for the warm-starts since they performed best in the general case and when restricting to contiguous solutions, respectively. We perform the warm-start using the solver's (CPLEX or CPOptimizer) starting point functionality to provide the values of the decision variables that correspond to the node sweeping schedule found by the heuristic. Other variables and the bounds on the objective value are left untouched.

6.1 Instance generation

We tested both on a set of random graphs and a set of random planar graphs. The former enabled us to test the models on instances with different structures, while the latter are closer to floor plans. Random graphs were generated using NetworkX [8] which allows the specification of number of nodes, n, and edges, m. We picked $m = \left\lceil p\frac{n(n-1)}{2} \right\rceil$ where p is the percentage of completeness of the generated graph. We generated a total of five graphs for every $n \in \{20, 30, 40\}$ and $p \in \{0.125, 0.25, 0.5, 0.75, 0.875\}$ (i.e., 75 instances in total). All graphs were tested for connectivity and non-connected graphs were replaced. The weights on the edges (resp., nodes) are picked uniformly at random between 1 and 4 robots (resp., between 2 and 10 robots for nodes). Node weight is constrained to be at least as high as the maximum weight of any incident edge. This is driven by the fact that many instances in the literature are made of rooms linked by narrower corridors and doors [12].

We generate random planar graphs using a linear-time algorithm based on Boltzmann samplers [6]. Since the generator does a uniform sampling of planar graphs, we generated as many graphs as needed to obtain a total of 20 connected instances for each $n \in \{20, 30, 40\}$. We used the procedure described above to assign the weights for each graph.

6.2 Results and discussion

Tables 1 and 2 show the average objective value, the number of times a method found the best incumbent solution, the number of times a method proved optimality, and the mean relative error (MRE), for random planar graphs and random graphs, respectively. The MRE is calculated to the best known lower bound of each problem instance, which it is either set to the optimal objective value (if found by any method) or to the highest lower bound reported by the MILP solver. The first section of the tables presents results in the general case whereas the second section shows results when enforcing contiguity. Models for which a warm-start is performed are suffixed by WS. We note that each method found a feasible solution on every instance.

Overall, the results empirically demonstrate the usefulness of optimization-based approaches for the GCP, particularly when warm-starting the solvers with a heuristic solution. Generally, the MILP model is the best in terms of finding and proving optimality,

[4] Although implemented in Python (which is in practice slower than C++), the implemented polynomial-time heuristics run in less than a second.

Table 1 Results summary for random planar graphs; the best result of each column is highlighted in bold on a per section basis

	Random planar graphs (60 instances)											
	Avg. Obj.			No. Best			No. Opt.			MRE (%)		
Method	20	30	40	20	30	40	20	30	40	20	30	40
GCP instances												
CPN	31.8	**37.0**	40.0	19	14	**10**	0	0	0	**0**	10	102
CPS	**31.6**	37.4	43.4	**20**	11	1	2	0	0	**0**	11	117
MILP	**31.6**	**37.0**	43.0	**20**	13	1	19	**1**	0	**0**	**9**	116
MaxSTH	38.4	48.7	51.5	0	0	0	–	–	–	21	45	160
MinSTH	50.1	66.0	72.4	0	0	0	–	–	–	58	95	265
CPN-WS	**31.6**	37.5	**39.5**	**20**	12	9	1	0	0	**0**	11	**98**
CPS-WS	**31.6**	37.4	40.4	**20**	12	**10**	2	0	0	**0**	11	102
MILP-WS	**31.6**	**37.0**	43.2	**20**	**15**	3	**20**	**1**	0	**0**	10	118
GCP instances with contiguity constraints												
CPN-C	31.8	37.5	40.0	19	14	**19**	0	0	0	**0**	22	181
CPS-C	31.8	**37.0**	42.8	18	**17**	2	2	0	0	**0**	**21**	201
MILP-C	**31.6**	37.1	56.9	**20**	16	0	**20**	0	0	**0**	22	312
MaxSTH-C	43.0	56.7	61.9	0	0	0	–	–	–	36	85	337
MinSTH-C	52.4	67.4	76.2	0	0	0	–	–	–	66	120	435
CPN-C-WS	**31.6**	37.2	**38.9**	**20**	16	18	1	0	0	**0**	**21**	**174**
CPS-C-WS	31.8	37.6	43.5	18	11	3	2	0	0	**0**	23	209
MILP-C-WS	**31.6**	**37.0**	48.2	**20**	16	1	**20**	**3**	0	**0**	**21**	248

while the CP models have the strongest performance *w.r.t.* average solution quality. Similar behavior is observed when we enforce the contiguity constraint. The results also confirm our hypothesis that using the maximum ST leads to a better heuristic than the minimum ST.

All methods produce solutions with large MRE, particularly on the larger problems. We suspect that a significant portion of this gap stems from weak dual bounds from the LP relaxations. Improvement to the dual bound is a key area for future work if we hope to solve these problems to optimality. Nonetheless, we note that the solutions found by our model-and-solve approaches outperform the various heuristics.

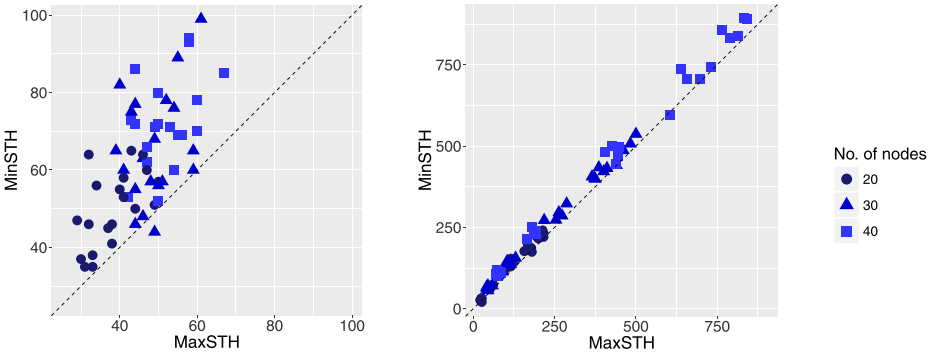
Delving deeper into solution quality, Fig. 5 compares the objective value obtained by MinSTH with the value obtained by MaxSTH. Results are presented for both random planar graphs and random graphs. A point represents an instance and its coordinates the objective value. Points below the dashed line indicate instances for which MinSTH found a higher quality solution. The MaxSTH heuristic outperforms the MinSTH heuristic on almost every instance, further confirming the use of maximum STs. Second, Fig. 6 compares the objective value obtained by MaxSTH with the value obtained by each of our models without warm-starting. The CP models outperform the heuristic on both types of graphs. In contrast, MILP performs worse than MaxSTH on larger random graphs, likely due to the large solution space and the weak LP relaxation.

Figure 7 shows the average of the ratios of the current incumbent objective value to the best objective value achieved by any of the four heuristics against the elapsed time in seconds. A ratio below one indicates that the method outperforms the best heuristic at

Table 2 Results summary for random graphs; the best result of each column is highlighted in bold on a per section basis

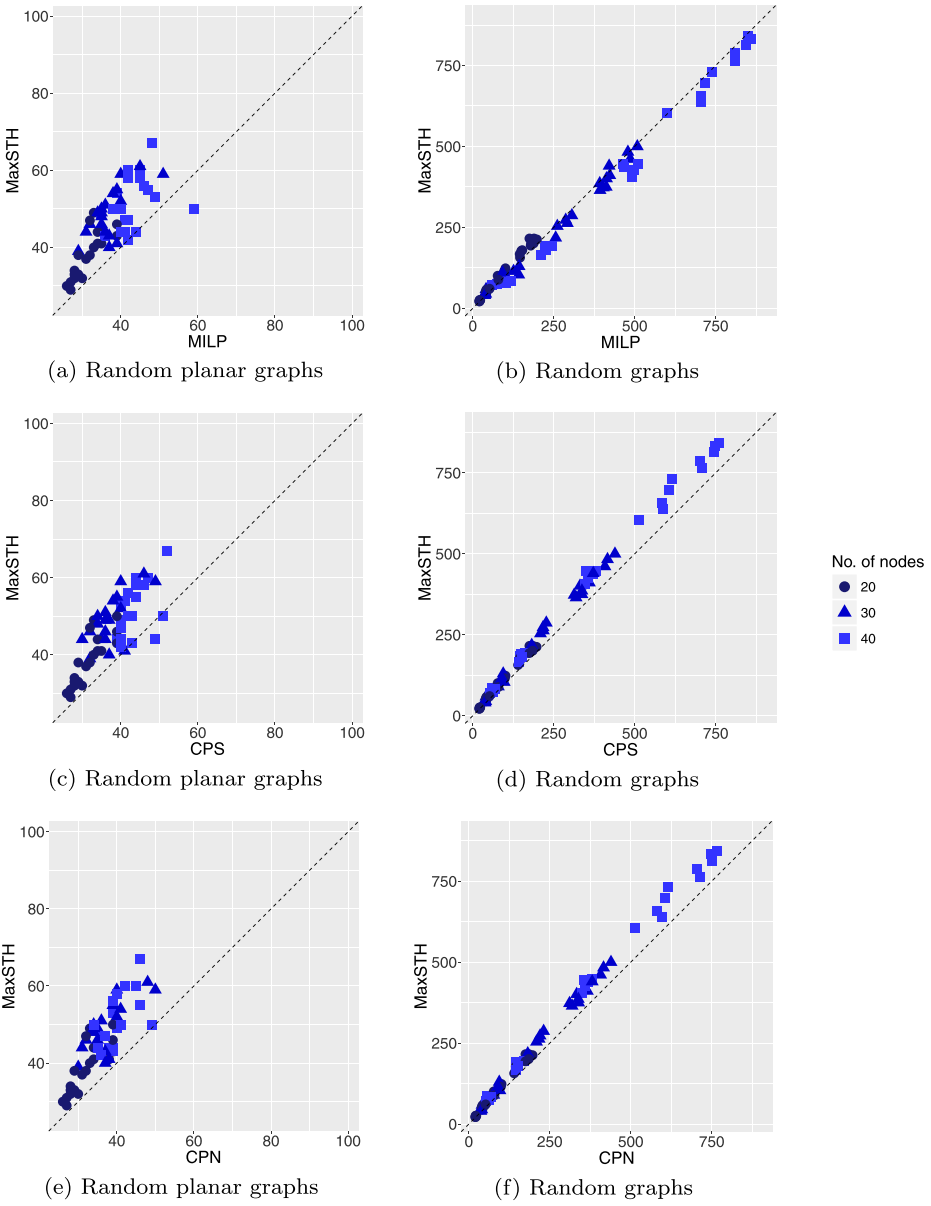
	Random graphs (75 instances)											
	Avg. Obj.			No. Best			No. Opt.			MRE (%)		
Method	20	30	40	20	30	40	20	30	40	20	30	40
GCP instances												
CPN	98.0	213.7	378.3	23	17	6	3	0	0	**28**	607	1027
CPS	98.0	213.0	377.1	24	17	8	0	0	0	**28**	604	1025
MILP	99.2	261.4	466.5	16	4	0	**12**	0	0	29	776	1382
MaxSTH	114.2	250.8	433.6	0	0	0	–	–	–	49	735	1224
MinSTH	129.6	277.3	476.8	1	0	0	–	–	–	72	827	1419
CPN-WS	98.0	213.6	**377.0**	24	**20**	10	3	0	0	**28**	605	**1023**
CPS-WS	**97.9**	**212.8**	377.5	**25**	19	**13**	0	0	0	**28**	**603**	1030
MILP-WS	98.9	247.4	431.4	19	5	1	10	0	0	29	724	1218
GCP instances with contiguity constraints												
CPN-C	**98.0**	214.4	379.6	24	**25**	**24**	3	0	0	**34**	654	1166
CPS-C	98.4	225.0	427.6	21	1	2	1	0	0	35	690	1328
MILP-C	99.0	259.3	471.6	18	3	0	**12**	0	0	35	826	1571
MaxSTH-C	115.1	253.0	439.4	0	0	0	–	–	–	60	797	1412
MinSTH-C	129.7	278.0	478.2	0	0	0	–	–	–	82	885	1602
CPN-C-WS	**98.0**	**213.4**	**377.0**	**25**	22	19	3	0	0	**34**	**650**	**1154**
CPS-C-WS	98.7	221.6	412.9	16	1	0	0	0	0	35	677	1276
MILP-C-WS	99.2	245.2	439.4	16	5	0	10	0	0	36	770	1412

a given point in time. We only present the first 30 seconds to emphasize the short time (usually less than 10 seconds) required by the CP models to outperform the best-performing heuristic. The CP methods clearly outperform the MILP approach within this time-frame. Further, we observe that CPN outperforms CPS on random planar graphs, but the converse holds for random graphs. It is unclear why this is the case.



(a) Random planar graphs (b) Random graphs

Fig. 5 The objective value obtained by MinSTH against the one obtained by MaxSTH for the random planar graphs (left) and the random graphs (right)



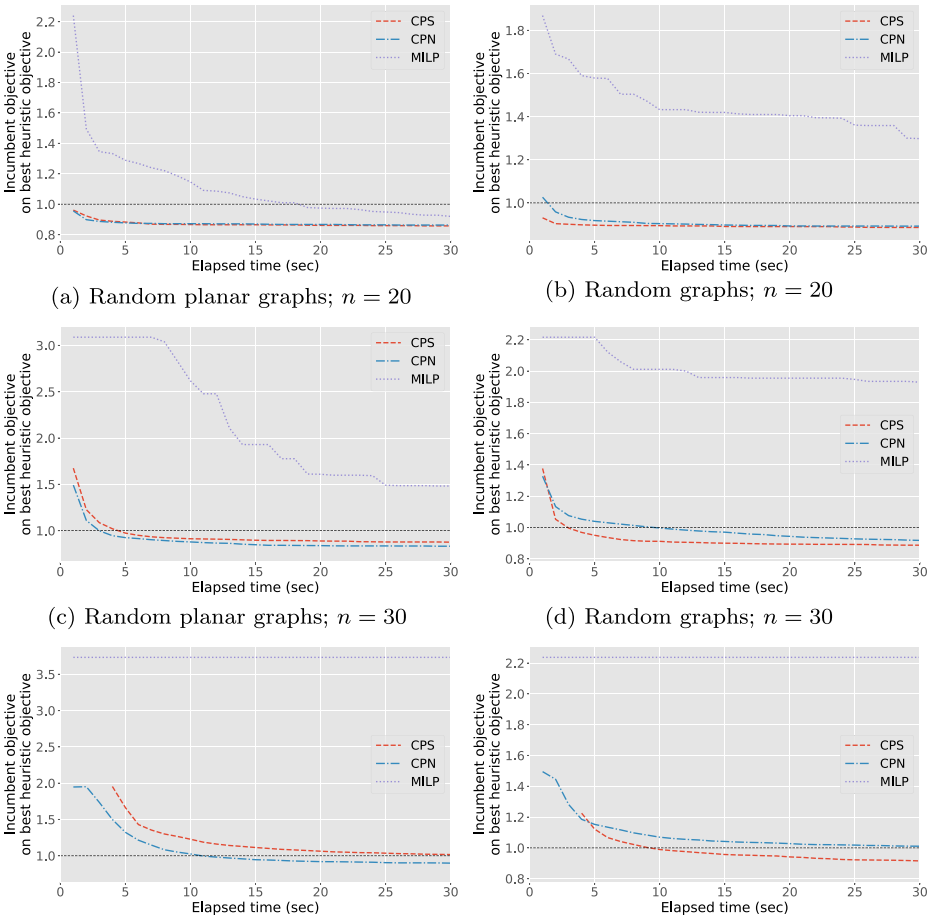
(a) Random planar graphs (b) Random graphs

(c) Random planar graphs (d) Random graphs

(e) Random planar graphs (f) Random graphs

Fig. 6 The objective value obtained by MaxSTH against the one obtained by each optimization-based approach within a 1-hour time limit for the random planar graphs (left) and the random graphs (right)

Finally, the warm-start results of Tables 1 and 2 confirm that the methods can be combined successfully with the heuristics. This technique yields an initial solution quickly (provided by the heuristic), while guaranteeing the completeness of the approach through the optimization-based model.



(a) Random planar graphs; $n = 20$

(b) Random graphs; $n = 20$

(c) Random planar graphs; $n = 30$

(d) Random graphs; $n = 30$

(e) Random planar graphs; $n = 40$

(f) Random graphs; $n = 40$

Fig. 7 The average of the per problem instance ratios of the current incumbent solution's objective value to the best objective attained by a heuristic for the random planar graphs (left) and the random graphs (right)

7 Conclusion

We investigated optimization-based techniques for the graph-clear problem (GCP), a variant of the pursuit-evasion problem, providing new formal results on progressive and contiguous assumptions and their impact on algorithm completeness. In particular, we proved that enforcing contiguity may remove all optimal solutions and, conversely, that preventing two special forms of recontamination does not remove all optimal solutions. However, the completeness for the general case of progressive solutions remains open.

We then presented mixed-integer linear programming and constraint programming models, and new heuristic variants to solve the GCP and compared them to previously proposed heuristics. Our empirical assessment on randomly generated problem instances indicates that our heuristic modifications improve upon the heuristics in the literature, that constraint

programming is able to find higher quality solutions than existing heuristics within runtimes reasonable for the application, and that mixed-integer linear programming is superior at proving optimality. We conclude that the proposed optimization-based techniques are currently the most suitable for solving the graph-clear problem.

In future work, we plan to extend our formal contributions to provide new results on the effect of partial recontamination, an open question. Additionally, we will investigate the use of decomposition approaches via a MILP/CP hybridization and develop GCP-specific branching heuristics for the CP models. Both the integration of MILP/CP technologies and problem-specific branching heuristics have been successful for other problems and represent promising strategies for improving on the methods we have presented.

Acknowledgements This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). M.P. Castro is funded by the Comisión Nacional de Investigación Científica y Tecnológica (CONICYT, Becas Chile). M. Morin is funded by the Fonds de Recherche du Québec – Nature et Technologies (FRQNT).

References

1. Barrière, L., Flocchini, P., Fraigniaud, P., Santoro, N. (2002). Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures* (pp. 200–209).
2. Beldiceanu, N., & Demassey, S. Global constraint catalog. http://sofdem.github.io/gccat/ (2014), accessed: 2017-11.
3. Booth, K.E.C., Nejat, G., Beck, J.C. (2016). A constraint programming approach to multi-robot task allocation and scheduling in retirement homes. In *International conference on principles and practice of constraint programming* (pp. 539–555): Springer.
4. Chung, T.H., Hollinger, G.A., Volkan, I. (2011). Search and pursuit-Evasion in mobile robotics. *Autonomous Robot*, *4*(31), 299–316.
5. Fomin, F.V., & Thilikos, D.M. (2008). An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, *399*(3), 236–245.
6. Fusy, E. (2009). Uniform random sampling of planar graphs in linear time. *Random Structures & Algorithms*, *35*(4), 464–522.
7. Garey, M.R., & Johnson, D.S. (1979). *Computers and intractability* Vol. 174. Freeman: San Francisco.
8. Hagberg, A., Swart, P., S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX. Tech. rep., Los Alamos National Laboratory (LANL).
9. Kirousis, L.M., & Papadimitriou, C.H. (1986). Searching and pebbling. *Theoretical Computer Science*, *47*, 205–218.
10. Kolling, A., & Carpin, S. (2007). Detecting intruders in complex environments with limited range mobile sensors. Robot Motion and Control, 417–425.
11. Kolling, A., & Carpin, S. (2007). The graph-clear problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1003–1008).
12. Kolling, A., & Carpin, S. (2008). Extracting surveillance graphs from robot maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2323–2328).
13. Kolling, A., & Carpin, S. (2008). Multi-robot surveillance: an improved algorithm for the graph-clear problem. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 2360–2365).
14. Kolling, A., & Carpin, S. (2010). Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, *26*(1), 32–47.
15. Kolling, A., & Carpin, S. (2010). Solving pursuit-evasion problems with graph-clear: an overview. In *Proceedings of the IEEE International Conference on Robotics and Automation. Workshop: Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees* (pp. 27–32).
16. Korsah, G.A., Kannan, B., Browning, B., Stentz, A., Dias, M.B. (2012). xBots: an approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 115–122).

17. Laurière, J.L. (1978). A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, *10*(1), 29–127.
18. Liu, J.N.K., Wang, M., Feng, B. (2005). IBOtguard: an internet-based intelligent robot security system using invariant face recognition against intruder. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *35*(1), 97–105.
19. Parker, L.E. (2002). Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous robots*, *12*(3), 231–255.
20. Parsons, T.D. (1978). Pursuit-evasion in a graph. In *Theory and applications of graphs* (pp. 426–441): Springer.
21. Qu, H., Kolling, A., Veres, S.M. (2014). Formulating robot pursuit-evasion strategies by model checking. *IFAC Proceedings*, *47*(3), 3048–3055.
22. Qu, H., Kolling, A., Veres, S.M. (2015). Computing time-optimal clearing strategies for pursuit-evasion problems with linear programming. In *Conference towards autonomous robotic systems* (pp. 216–228): Springer.
23. Shimosasa, Y., Kanemoto, J., Hakamada, K., Horii, H., Ariki, T., Sugawara, Y., Kojio, F., Kimura, A., Yuta, S. (1999). Security service system using autonomous mobile robot. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, (Vol. 4 pp. 825–829).
24. Van Hentenryck, P., & Carillon, J.P. (1988). Generality versus specificity: an experience with AI and OR techniques. In *National conference on artificial intelligence (AAAI)* (pp. 660–664).