Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

A spatially explicit evolutionary algorithm for the spatial partitioning problem[☆]

Yan Y. Liu [a,*,1], Wendy K. Tam Cho [b]

[a] Computational Urban Sciences Group of the Computational Sciences and Engineering Division at Oak Ridge National Laboratory, 1 Bethal Valley, P.O. Box 2008, Oak Ridge, TN 37830, United States of America

[b] Departments of Political Science, Statistics, Mathematics, and Asian American Studies, the College of Law, and the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, 420 David Kinley Hall, 1407 W. Gregory St., Urbana, IL 61801, United States of America

ARTICLE INFO

Article history:
Received 30 May 2019
Received in revised form 20 October 2019
Accepted 22 January 2020
Available online 4 February 2020

Keywords:
Combinatorial optimization
Evolutionary algorithm
Spatial optimization
Heuristics
Parallel computing

ABSTRACT

Spatial optimization seeks optimal allocation or arrangement of spatial units under constraints such as distance, adjacency, contiguity, and pattern. Evolutionary Algorithms (EAs) are well-known optimization heuristics. However, classic EAs, based on a binary problem encoding and bit-operation-based offspring operators, are spatially unaware and do not capture topological and geometric relationships. Unsurprisingly when spatial characteristics are not explicitly considered in the design of EA operators, that EA becomes ineffective because satisfying spatial constraints is computationally expensive. We design and develop novel spatially explicit EA recombination operators, inspired by the path relinking and ejection chain heuristic strategies, that implement crossover and mutation using intelligently guided strategies in a spatially constrained decision space. Our spatial EA approach is general and slots well into the foundational theory of evolutionary algorithms for spatial optimization. We demonstrate improved solution quality and computational performance with a large-scale spatial partitioning application.

1. Introduction

Spatial optimization problems seek optimal allocation or arrangement of spatial units according to some objective or measure of goodness [1]. The spatial properties embedded in the objectives and constraints may include, but are not limited to distance, adjacency, contiguity, containment, intersection, shape, partition, and pattern [2]. Spatial optimization has been studied in a variety of contexts, dating back to the 1800s with explorations of efficient land use activity [3]. More recent research includes studies of location models [4–7], coverage problems [8–10], zoning and spatial aggregation [11,12], and spatial characteristics in land cover classification [13–15]. To be sure, research in spatial optimization is lively and generates significant technical and substantive interest across an array of different areas.

Many problems in spatial optimization are computationally challenging and even computationally intractable (i.e., *NP*-Hard [16,17]). Examples include the regionalization problem that aims to group spatial units into a small set of regions that satisfy optimization objectives (e.g., the *p*-region problem [18] and the spatial clustering-based regionalization [12]), the spatial zoning problem that seeks to partition spatial units into a set of contiguous zones [19], the maximal covering location problem where the coverage of a set of facilities is maximized [8], the spatial allocation problem that seeks to allocate specific activities to particular spatial units [20], and the *p*-hub location problem that locates *p* transportation hubs and allocates demand to specific hubs to minimize total transportation costs [21]. With the increased size and dimension of spatial data, the associated spatial optimization problem becomes more intricate, characterized by massive decision spaces that eclipse the capabilities of exact algorithms to identify optimal solutions. Given these trends, continued success in deploying future spatial optimization applications requires innovation in spatial optimization methods.

☆ This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

* Corresponding author.
E-mail addresses: yanliu@ornl.gov (Y.Y. Liu), wendycho@illinois.edu (W.K.T. Cho).

1 Liu's work is supported in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy under contract DE-AC05-00OR22725.

https://doi.org/10.1016/j.asoc.2020.106129
1568-4946/© 2020 Elsevier B.V. All rights reserved.

One path for solving difficult optimization problems is via heuristics and soft computing techniques such as Evolutionary Algorithms (EAs), which are well-known metaheuristics inspired by natural selection [22]. EAs mimic an evolutionary process that encodes a problem as a *chromosome* and generates an initial population of individual solutions with random chromosomes. Through randomized EA operators (e.g., selection, crossover, mutation, and replacement), the population evolves based on a "survival of the fittest" rule [23,24]. This route has been successful for many large-scale optimization problems. However, complex spatial optimization problems pose interesting and non-trivial challenges, necessitating novel strategies that incorporate spatial elements into the EA operators [25].

Xiao [1] categorized problems in spatial optimization as either assignment or partition problems, with or without spatial constraints. Among the four categories, we investigate partition problems with spatial constraints such as contiguity (hereafter *spatial partitioning*), where arriving at different partitions necessarily involves reassignment of neighboring units that satisfies the contiguity constraint. A particular search complication that arises in these partitioning problems is that considerations of spatially defined locality and adjacency have a direct and explicit effect on solution feasibility. While non-spatial set partitioning problems involve the unconstrained combinatorial construction of sets, the system of spatial dependency among the decision variables in spatial partitioning redefines solution feasibility. One consequence is that the decision space becomes "patchy", with feasible and infeasible solution areas juxtaposed. In addition, as the problem size grows, the disparity between the number of feasible and infeasible partitions grows, with infeasible partitions quickly eclipsing the feasible ones.

Although randomized components powerfully enable EA performance, the classic operations that generate new solutions by exchanging bits on linearly-encoded chromosome are ineffective in traversing spatially-constrained decision spaces because they do not capture or preserve the variable interdependencies introduced by the spatial constraints [26]. This issue becomes even more poignant for large applications that have increasingly disparate proportions of infeasible solutions. In addition, conventional strategies such as designing a penalty function for spatial constraint handling are also unlikely to be effective since they expend enormous computational effort wading into infeasible regions rather than avoiding them all together.

While it is possible, it would be highly unusual that a spatial partitioning problem could be solved by conventional EAs with linear recombination operators [25]. One possible route is to use the classic linearly-coded non-spatial operators to generate new solutions, and then to subsequently repair the broken spatial relationships. While this strategy may work on small problem instances, it becomes computationally prohibitive as the problem size increases and thus devolves into a random search for feasible solutions rather than a heuristic that improves solution quality. We argue that *EA recombination operators, including crossover and mutation, must not only be spatially aware, but also spatially explicit in exploring the spatially constrained decision space*. Such intelligently guided constructive methods are more promising than repair methods, particularly for large problem sizes.

We propose a novel spatial evolutionary algorithm that implements spatially explicit crossover and mutation operations via an adaptation of the path relinking and ejection chain heuristics. Our proposed EA preserves solution feasibility and thus reduces wasted computation exploring infeasible solution regions. Consequently, our EA is able to improve solution quality on large problem instances and maintain desirable computational performance.

Since spatial heuristics are incorporated into the EA operators, the algorithm falls into the class of memetic algorithms [27] or hybrid EAs [28]. Our computational approach also falls within the realm of soft computing that involves the combination of multiple methods (often involving evolutionary computing or swarm intelligence algorithms) that each bring a helpful element for achieving a complex goal [29]. Soft computing approaches have driven advances across a variety of different scientific domains, including portfolio optimization, rainfall prediction, traffic flow, and image processing [30–34]. Our approach is developed based on a general definition of the spatial partitioning problem. We solve a specific spatial zoning application to demonstrate the algorithmic and computational performance of the proposed approach with both sequential and parallel implementations.

2. Literature review

The EA approach to spatial optimization has been an active area of research in geography for decades. Although EAs are recognized as a general heuristic for problem-solving and have been successfully employed in various spatial optimization problems [35,36], how to align EA encoding and operations with an application's inherent data idiosyncrasies [37] has remained a challenge for achieving desirable performance. For example, Hosage and Goodchild [25] found that the computational efficiency of EAs lags other algorithms for solving the *p*-median problem, commenting that "one would expect somewhat different performance if the coding order preserved spatial relationships than if it were essentially random".

Adapting heuristics to function efficiently with spatial data has been a consistent effort in geographic analysis research. Openshaw [38] first introduced a contiguity-preserving method that generates new solutions by morphing only on zone boundaries to study geospatial partitioning applications. Based on this method, Openshaw and Rao [39] developed tabu and simulated annealing heuristic methods for census zoning. Tong et al. [40] implemented a spatial genetic algorithm for the maximum coverage problem with a specialized crossover procedure that avoids spatial clusters and promotes spatial dispersion. Xiao et al. [41] designed a spatial multi-objective EA for the site search problem by devising location-based operators to move a subset of units to a randomly identified location while morph-based operators identified moveable units between contiguous sites and "morphed" their site assignment. Various heuristics have been developed for spatial optimization by the geocomputation community [42].

In these applications, one of three main strategies has been employed to enforce spatial contiguity. First, solutions are generated freely without spatial considerations. The non-contiguous solutions are then discarded [43,44]. While this strategy is realizable, it is impractical for large applications where non-contiguous solutions far outnumber contiguous ones. Second, one generates solutions freely, and then repairs the broken spatial relationships. Again, while this strategy may work on small problem instances, it becomes computationally prohibitive as the problem size increases. Lastly, contiguity can be enforced by creating only contiguous aggregations. This approach is the most promising and has been explored by a number of scholars. Cova and Church [45] maintain contiguity via a series of math inequalities using 1D and 2D hamming distances to represent the principle that no spatial unit can be chosen before a unit closer to a reference unit is chosen. Cleverly, the hamming distance is defined through a directional spatial relation between a cell and its neighbors toward the reference site. This representation permits a mixed integer programming (MIP) system to make implicit spatial neighborhood moves via branching that is based on distance. Using a similar representation, Williams [46] addressed the exact contiguity constraint for a vector-based problem formulation. Shirabe

[47] then optimized the contiguity representation for integer programming in a way that reduces the search cost for linear solvers. Shirabe [48] and Murray et al. [49] conceptualized unit connectivity as a network and defined contiguity to be a classic network flow problem, in which a connected graph allows the selection of paths from source to sink.

While a mathematical programming solution to contiguity-constrained spatial partitioning needs to address how to best represent and incorporate the contiguity constraint, heuristic solutions need to consider contiguity explicitly in neighborhood search functions. In this quest, one could introduce a penalty function as a general solution, though as Runarsson and Yao [50] point out, a penalty function is both hard to define and to make efficient. Instead, existing efforts devise specialized routines to satisfy contiguity. For example, in their spatial scanner that searches for clusters in a map, Izakian and Pedrycz [51] designed a particle swarm optimization algorithm that checked the contiguity of *each* cluster. King et al. [52] designed a specialized geo-graph data structure. While their formulation improved a single contiguity check, overall improvement was limited because of the significant computational performance penalty imposed by the need to perform this check at *every* EA iteration. Indeed, Liu et al. [19] improved efficiency by two orders of magnitude by forcing contiguity in initial solutions and requiring contiguity checking only in solution updates.

To be sure, our goal is not simply to incorporate spatially aware operators, but to do so in a computationally efficient manner that will scale with problem size. In this pursuit, scalability is most promisingly fostered by avoiding spatially infeasible solutions with carefully designed operators that preserve spatial constraints.

3. Methodology

Our main observation is that a failure to recognize and incorporate spatial relationships and constraints significantly degrades the performance of EA operators. In this section, we start with a mixed integer programming (MIP) formulation of a generalized form of the spatial partitioning problem constrained by contiguity. We highlight how the lack of spatial awareness in conventional EA operators creates computational issues. An avenue to overcome these limitations is then proposed by devising new EA operators that incorporate spatially explicit operations.

The Specification of the Spatial Partitioning Problem

Problem Statement

Given n spatial units, partition the n units into k disjoint zones that minimize a defined objective function and satisfy a set of spatial and non-spatial constraints, in which the contiguity constraint requires all k zones and all of the units in each of the k zones to be connected.

A Mathematical Formulation

I: set of spatial units;

A: set of adjacent unit pairs;

K: set of zones;

n_k: number of units in zone k;

$$\mathbf{x} = \{x_{ik}\} : x_{ik} = \begin{cases} 1 & \text{if unit } i \text{ is assigned to zone } k \\ 0 & \text{otherwise} \end{cases}$$

y_{ijk}: flow from unit i to unit j for zone k

$$w_{ik} = \begin{cases} 1 & \text{if unit } i \text{ is the hub of zone } k \\ 0 & \text{otherwise} \end{cases}$$

Objective: $\min f(\mathbf{x})$

Constraints:

$$\sum_{j\,|\,(i,j)\in A} y_{ijk} - \sum_{j\,|\,(j,i)\in A} y_{jik} = n_k w_{ik} - x_{ik} \qquad \forall k \in K,\ \forall i \in I \tag{1}$$

$$\sum_{j\,|\,(j,i)\in A} y_{jik} \leqslant (n_k - 1)x_{ik} \qquad \forall k \in K,\ \forall i \in I \tag{2}$$

$$\sum_{k\in K} x_{ik} = 1 \qquad \forall i \in I \tag{3}$$

$$\sum_{i\in i} w_{ik} = 1 \qquad \forall k \in K \tag{4}$$

$$\mathbf{ax} \leqslant b \tag{5}$$

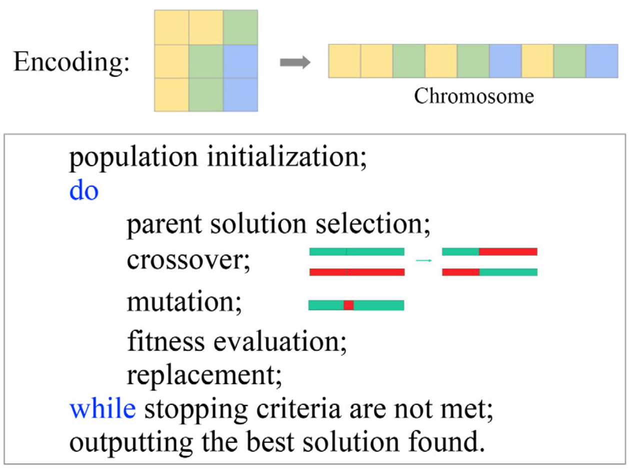
$$x_{ik}, w_{ik} \in \{0, 1\} \qquad \forall k \in K, \forall i \in I \tag{6}$$

$$y_{ijk} \geqslant 0 \qquad \forall k \in K, \forall (i, j) \in A \tag{7}$$

The above MIP formulation is a variation of Shirabe [48], which defines the contiguity for each partition as a network flow from all of the spatial units in each zone to their zone hub that receives the flow. The objective function is a weighted sum of spatial and non-spatial objectives. For instance, if the partitioning problem considers weight balancing among zones, the objective function can minimize the weight difference. Constraint (1) requires that the difference of flow into and out of a unit i must be $(n_k - 1)$. This means that if unit i is the hub of zone k, the flow traverses each unit in the zone exactly once. Otherwise, this constraint has no effect. Constraint (2) ensures that no unit is visited twice. Constraint (3) guarantees that each unit is a member of one and only one zone. Constraint (4) guarantees that each zone has only one hub. These four constraints together ensure that all of the units are partitioned into exactly k contiguous zones. Constraint (5) encompasses all other non-spatial constraints. While unit assignment is discrete (Constraint (6)), the flow is formulated as a continuous variable (Constraint (7)). We can also see that this problem is computationally intractable since Constraints (1) and (2) generate a number of inequalities that increases exponentially with the number of units.

It is worth clarifying that the spatial partitioning problem is similar to, but distinct from, both the graph partitioning problem as well as the k-means clustering problem [16]. While the graph partitioning problem also seeks to balance aggregated weights among the k partitions, it also requires the edge weights across the partitions to be minimized, but does not enforce contiguity. Spatial partitioning is also similar to the k-means clustering problem [53], but the decision space search is based on the adjacency graph, instead of on units and distance.

The above network flow abstraction provides a mathematical interface for an MIP solver. For heuristic solutions, however, such a flow definition is not necessary because the contiguity constraint can be handled with a penalty function or enforced within the neighborhood search routine. An EA solver typically encodes a partition solution into a linear array that is indexed by spatial units, $\{x_i \,|\, x_i \in I\}$, where the zone index is the value of each element. A fitness function captures the objective function. If a penalty function is defined to handle infeasible solutions that violate any of the constraints, the fitness function would be different from the objective function. Alternatively, the fitness and objective functions can be identical, and one can define a separate unfitness function. New solutions can then be generated using the crossover and mutation operators in each EA iteration. The evaluation of new solutions checks all of the constraints and returns a score for each solution. A replacement strategy uses this score (and the value of the unfitness function, if defined) to select a subset of the population for possible replacement. Fig. 1



```
population initialization;
do
    parent solution selection;
    crossover;
    mutation;
    fitness evaluation;
    replacement;
while stopping criteria are not met;
outputting the best solution found.
```

Fig. 1. A general evolutionary algorithm (EA) for spatial partitioning.

illustrates a general EA approach. The rest of the paper describes the limitations of the conventional crossover and mutation in solving the spatial partitioning problem and proposes new spatial crossover and mutation operators.

3.1. *Satisfying spatial constraints*

We begin with a simple illustration to demonstrate how a contiguity requirement alters the search of the decision space for the classic 1-bit EA mutation operator. For ease of illustration, we use a raster representation of the spatial variables. Our proposed algorithm applies to both vector- and raster-based problems since we adopt an adjacency graph structure to represent neighborhood, which generalizes the adjacency of spatial objects of both regular (e.g., raster) and irregular shapes. In this example, the adjacency is rook-based, which means that if two units are connected by only a point (queen connectivity), then they are not considered connected.

In Fig. 2, the leftmost figure shows nine spatial units partitioned into three contiguous zones. Using a classic 1-bit mutation, the new solutions that result from mutating a single cell are shown in the middle figure. The number of possibilities is $2 \times 9 = 18$, because each cell can be re-assigned to either of the other two zones. However, note that mutating cell $(1, 1)$ breaks contiguity and necessitates a repair operator to restore contiguity. The rightmost figure in Fig. 2 demonstrates a repair scenario. First, cell $(1, 2)$ switches from yellow to blue. The repair operator then connects it to the existing blue zone, $\{(3, 3)\}$, by identifying a route between them and re-assigning the cells on that route to the blue zone while maintaining the other two zones. The arrows show possible repair paths. Note that the route $(1, 1) \rightarrow (2, 1) \rightarrow (3, 1) \rightarrow (3, 2)$ is not possible because it eliminates the yellow zone. Similarly, the route $(2, 2) \rightarrow (2, 3)$ is not possible because it splits the green zone. As we can see, repairing non-contiguous solutions is computationally costly, even for a simple mutation operator.

3.2. *Limitations of EA crossover operators*

Classic linear EA operators have two main drawbacks. First, even in an unconstrained decision space, the search space derived from linear recombination is smaller than the enumerable possibilities [54]. Hence, these operators limit and may miss feasible search regions. Second, since these recombination operators are spatially unaware, they may generate solutions that violate the spatial constraints. These solutions would either be discarded, leading to poor efficiency, or would be repaired, which is non-trivial and computationally expensive.

Fig. 3 illustrates these issues with two commonly employed crossover operators. The first, the classic linear crossover using a single *cutpoint*, is shown in the upper diagram of Fig. 3. Here, the two parent solutions are shown on the left as 2D and 1D encoded (chromosome) views. After a cutpoint is chosen, the crossover generates two new solutions, shown on the right, by swapping the second part of one chromosome and attaching it to the first part of the other chromosome. However, since neither of the two new solutions is contiguous, they are both infeasible.

The bottom diagram in Fig. 3 illustrates a spatially-aware crossover operator that overlaps two solutions and randomly reassigns the resulting subdivisions into contiguous zones [1]. A subdivision is identified with a unique $\langle zone1, zone2 \rangle$ label which denotes a cell's zone assignment from the two parent solutions. Ideally, the favorable attributes from both parent solutions are preserved, though this is not guaranteed as we can see from the subdivisions shown in the diagram. In the extreme case, the overlap creates 9 subdivisions, which results in no progress since the new problem is identical in complexity to the original problem. Moreover, it is not clear how a random graph cut can be specified to preserve and propagate favorable subdivisions. Despite these drawbacks, however, a significant benefit of the overlap crossover remains—it can create a search space of size at least $3^3 - 2 = 25$ (by taking the largest three subdivisions as starting zones and parceling out the remaining three cells, but subtracting the two existing solutions), which is a significant improvement over the classic EA crossover.

3.3. *Spatial recombination approach*

These examples highlight that an important component has been missing from the extant EA *spatial recombination* operators. In particular, they do not consider spatial characteristics *during* the new solution generation phase. Spatial recombination operators must transform the spatial characteristics into quantifiable measures that direct the decision space search. This is akin to incorporating domain knowledge into heuristics. The main difference is that spatial characteristics require an integrated framework for incorporating spatial elements while maintaining the efficiency of each EA iteration. In this direction, Xiao [1] made an initial effort by integrating graph theoretic components to categorize spatial optimization problems. This effort needs to be further extended, and more importantly, must be systematic.

We propose a *Spatially Explicit Evolutionary Computation* (SEEC) approach that adheres to the basic structure of EA recombination procedures and scales to large problem instances. We represent spatial relationships using graph data structures, on which we design a chain of local moves that collectively comprise a large disturbance in the search neighborhood. Chained moves are desirable since a chain, which is tied on either end to parent solutions, allows one to design globally large, but locally incremental moves, within the search space. The length of the chain may be adjusted to permit a controlled but sufficiently large move to make an impact. Auspiciously, chained moves can be seen as a generalization of the mutation and crossover operators in EAs. A mutation operator can be designed as a series of chained moves where the chain, at both ends, is anchored to a single parent solution. A crossover operator can be designed as a set of chained moves that comprise a "walk" from one parent solution to a different parent solution. The movement along this chain generates a series of intermediate or child solutions.

Coupling randomization with a chaining mechanism provides a constructive method for designing spatial recombination strategies that incorporate spatial neighborhoods and chaining on

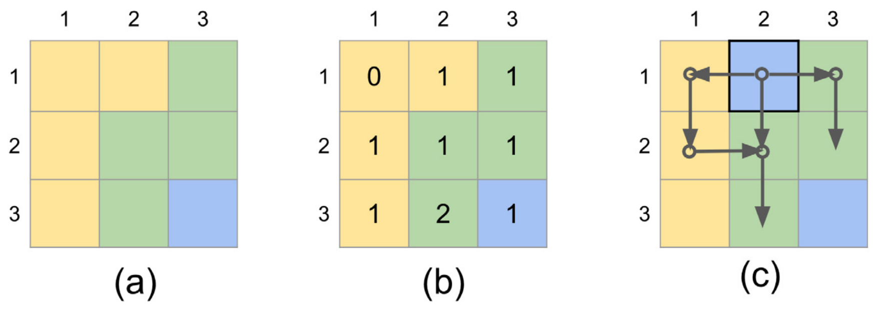


Fig. 2. Search Space of Mutation for Spatial Optimization Problems (in a Rook Neighborhood). Subfigure (a) is a solution for 9 spatial variables on a 2D raster layout. Subfigure (b) shows the number of mutable possibilities for each cell. Subfigure (c) illustrates search paths needed to repair a mutation operation.

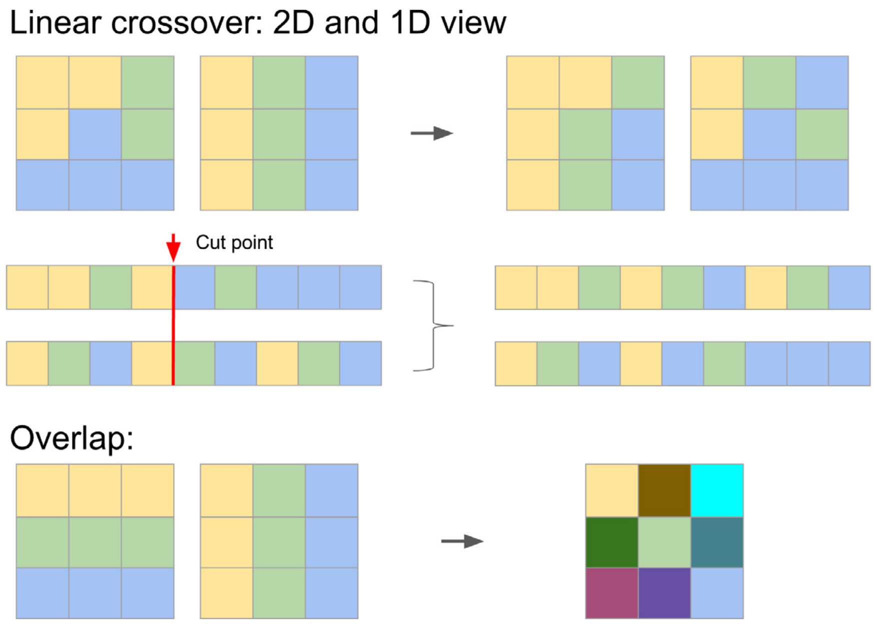


Fig. 3. Limitations of a Classic EA Crossover. The upper diagrams illustrate a linear recombination crossover. The lower diagram illustrates an overlap-based crossover.

graphs. This observation inspires the design of the spatial crossover and mutation operators in this paper, which were inspired by two well-established heuristic methods, path relinking and ejection chain. Detailed description of these two methods in an operations research context can be found in [55,56], both of which embody the chaining concept to efficiently and effectively solve large combinatorial (non-spatial) optimization problems.

3.4. Adapting path relinking and ejection chain

A path relinking process begins by identifying two solutions, an initial (or source) solution, S, and a reference (or target) solution, T. A path that links these two solutions transforms the source solution to the target solution. The hope is that somewhere along the path (in the neighboring space) lie new and better solutions that are a mix of elements found in S and T. These moves can be designed adaptively to improve the performance of the heuristic. The path links multiple random and purposeful moves, where each move is an incremental change from the previous state. Path length is a function of solution distance from S to T, which can be defined for non-spatial problems as the number of variables where the two parent solutions have different values. At each step, a neighborhood function randomly identifies a local move. With each step forward, the distance is reduced along the relinking path, reaching zero when the relinking is complete. The spatial adaptation of path relinking as the crossover operator in EA is illustrated in Fig. 4.

An ejection chain consists of a series of moves originating from one solution to generate multiple new solutions. Each move "ejects" an assignment of a problem variable to the next variable. This type of ejection forms a chain that can be either cyclic or acyclic. An ejection chain is a generalization of local neighborhood functions such as shift (chain length = 1) and swap (chain length = 2). Non-spatial or spatial considerations may be utilized to determine the chain length and the units to eject in each move.

Adapting general path relinking and ejection chain principles into a spatial context is not straightforward and must incorporate various strategies. Solution distance, which is often defined as the number of alleles with different values between two solutions, is not applicable in spatial configurations because spatial partitions differ in their partition shape. Zones with the same shape but different zone indices are identical but would be considered different in a chromosome encoding. For this reason, while a path relinking process can surely transform one chromosome to another with bit exchanges, spatial constraints such as contiguity, containment, or shape, may shorten, lengthen, or even block the path. The design of spatial recombination operators is, thus, significantly more complex than the classic crossover and mutation, or conventional path relinking and basic ejection chain methods.

3.5. Spatial crossover through path relinking

Fig. 4 shows a simple partitioning problem with a contiguity constraint and illustrates the steps in our path relinking-based

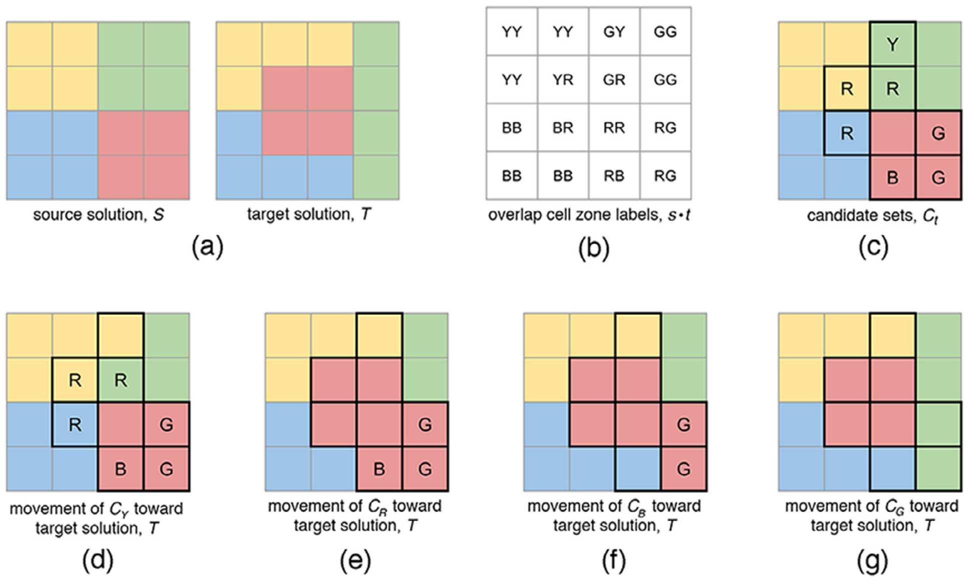

source solution, S
target solution, T
(a)
YY YY GY GG
YY YR GR GG
BB BR RR RG
BB BB RB RG
overlap cell zone labels, s•t
(b)
Y R R R G B G
candidate sets, C_t
(c)
R R R G B G
movement of C_Y toward target solution, T
(d)
G B G
movement of C_R toward target solution, T
(e)
G G
movement of C_B toward target solution, T
(f)
movement of C_G toward target solution, T
(g)

Fig. 4. Illustration of PRCRX, a spatial path relinking-based crossover operator.

crossover (PRCRX) algorithm. The example problem partitions 4 contiguous zones on a 4 × 4 grid, where each cell is a problem variable. Subfigure (a) shows two parent solutions. The solution on the left is the *source solution*, S, while the solution on the right is the *target solution*, T. The colors represent different zones. We first "overlap" these two solutions (shown in subfigure (b)), and assign a *zone label*, $s \bullet t$, to each cell, where s is the cell's zone assignment (here, representing the cell color) in S, and t is the zone assignment in T. Adjacent cells with the same zone label form a group or set of connected components. In the illustration, there are 10 groups (YY, GY, GG, YR, GR, BB, BR, RR, RG, and RB). Adjacent groups will have different zone labels, while non-adjacent groups may have the same zone label.

For the k-partition problem, we next pick k seed groups, where each group, G_t, has a unique target zone label, t. If they exist, we could simply pick the groups with zone label $z \bullet z$, for $z = 1, \ldots, k$. The solution distance is defined to be the difference between the total number of cells and the number of cells in the k seed groups. Here, since there are 16 total cells and 9 cells in the 4 seed groups, (3 YY, 2 GG, 3 BB, 1 RR), the solution distance is $16 - 9 = 7$. These 7 cells (shown with their target solution label, t, and outlined with a black border in Fig. 4(c)), will mutate from their zone in the source solution to their zone in the target solution on the "walk" from the source solution to the target solution. The cells with a common label, t, form a *candidate set*, C_t. Here, there are 4 candidate sets, C_Y (consisting of the cell with label GY), C_R (consisting of the 3 cells with labels YR, GR, or BR), C_G (consisting of the 2 cells with label RG), and C_B (consisting of the cell with label RB). For any zone label, t, the union of the candidate set, C_t and the seed group, G_t, is the set of all cells in the target solution with label t, i.e., $C_t \cup G_t = T_t$. In Fig. 4(c), we can quickly see the cardinality of the candidate sets: $|C_Y| = 1$, $|C_R| = 3$, $|C_B| = 1$, $|C_G| = 2$.

Definition. The **solution distance**, d, between the source solution, S, and the target solution, T, is defined as $d = \sum_{z=1}^{k} |C_z| = \sum_{z=1}^{k} |T_z - G_z|$.

We may begin the relinking process between the two solutions by first choosing a particular source zone, i.e. a set of units that share the same zone label, s. In each *move*, or step of the path, if a cell is converted from zone s to zone t, the solution distance, d, is reduced by 1. Fig. 4(d–f) provide an example of a path relinking process that morphs the source solution to the target solution. To move from subfigure (c) to subfigure (d), we convert all of the cells labeled "Y" to the yellow zone. To move to the solution shown in subfigure (e), the cells labeled "R" are converted to the red zone. To obtain subfigure (f), the cells labeled "B" are converted to the blue zone. Finally, we convert the cells labeled "G" to the green zone, which completes the path. Here, this path of length 4 travels a distance of 7 and generates 4 intermediate solutions.

Notice that there are many ways in which the path may be constructed. One could, alternatively, move each candidate cell individually, generating 7 intermediate solutions (6 new). Or, as long as each move does not violate the spatial constraints, we could build a path with varying step sizes. The *path length* is flexible, but constrained by the number of available mutable candidate cells. It is possible that a particular order of candidate set/unit visits may not be viable, but this does not pose an issue since the purpose of the walk is to generate intermediate solutions, not necessarily to end at solution T.

In the construction of any path, an important consideration is to properly deliberate seed groups before the relinking process begins to avoid the unnecessary computation that arises from generating infeasible intermediate solutions. For example, recall that when the k seed groups were created, we *chose* those groups with identical s and t zone labels. This is one of a set of possible choices. We could have chosen any $s \bullet t$ label where the k seed groups have k unique s and k unique t labels. However, as Fig. 5 illustrates, there is not always a unique zone labeling in the overlap solution. If a unique zone labeling for both S and T does not exist, the two source zones will merge as we expand the groups, G_t, because there are at least two seed groups that share a source zone. Moreover, in the expansion of the two seed groups to target zones, the fitness of the intermediate solutions is likely to be worse. Accordingly, in our implementation, we seek to identify unique zone labelings and to minimize groups with the same source zone labels. Another complication that may arise is that a violation of the contiguity constraint may prevent the completion of a path. Fig. 6 illustrates that if a group, G_t, or units in C_t are poorly selected, it may be impossible to expand G_t.

It is difficult to know a priori whether a seed grouping or a particular selection will lead to one of these problematic scenarios. This phenomenon affects the search space size, but the effect

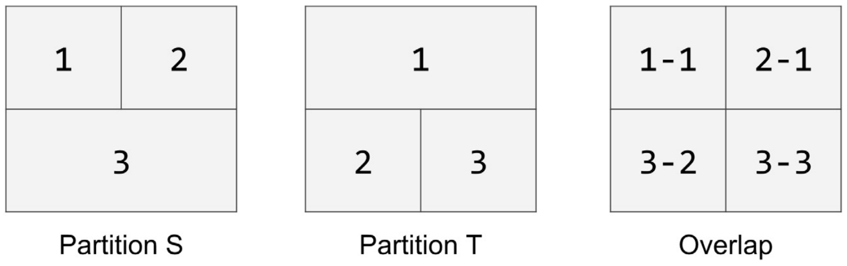


Fig. 5. Considerations for Zone Labeling. Two seed groupings with unique *t* labels are possible: (1 • 1, 3 • 2, 3 • 3) or (2 • 1, 3 • 2, 3 • 3). However, since neither has unique *S* labels, the expansion to *T* zones with any of the two seed groupings has two undesirable effects. First, zone 1 and 2 in partition *S* will eventually merge unless the expansion is terminated early. Second, zone 3 in *S* cannot be expanded or inappropriately expands both zone 2 and 3 in *T*.

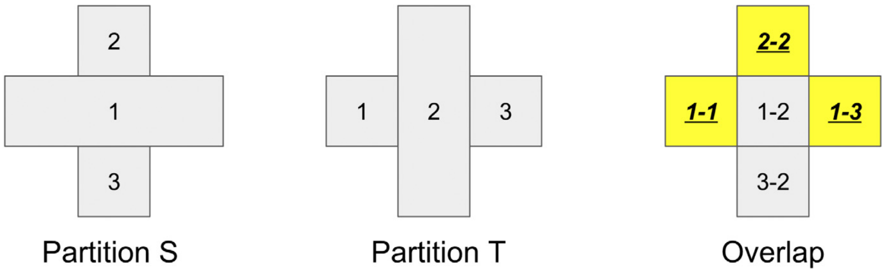


Fig. 6. Seed Group Selection Constrained by Source Zone Contiguity. The selection of the yellow seed groups makes it impossible to maintain the contiguity of source zone 1. The expansion of group $G_2(2 \bullet 2)$ to group $G_2(1 \bullet 2)$ disconnects source zone 1. An appropriate zone labeling would select $G_2(1 \bullet 2)$ as the seed group.

on fitness improvement is unknown. Furthermore, sophisticated algorithms for preventing such scenarios may be too costly to compute. Our current implementation detects and avoids moves that violate the contiguity constraint. If these are all the possible candidate moves, the relinking process is simply terminated. Other implementations are possible and can be flexibly designed.

3.6. Spatial mutation through ejection chain

ECMUT (ejection chain-based mutation) builds on a previous spatial mutation operator [19] by generalizing the chained mutation steps into the ejection chain heuristic framework [57]. To transform a general ejection chain to one that is spatially explicit, we need to specify a spatial neighborhood function, which could be, for example, defined by a traversal that proceeds along adjacent vertices.

Notice that ejection chain-based mutation has an advantage over the location-based morph spatial design [1]. The drawback of the latter is that, while the new area is able to contain the shape of the units, variable interdependencies are not preserved. Ejection chain-based mutation, on the other hand, preserves spatial dependencies since the exchange of unit assignment occurs among neighboring units.

Fig. 7 provides an illustration of the ECMUT process. Here, the ejection chain is cyclic, with the cycle (yellow → blue → red → green →), which returns to yellow after green. Subfigure (a) shows a parent solution with 4 colored zones. At each step of the chain, a random set of contiguous cells from one zone is moved or "ejected" to its neighboring zone. Since the zone order is yellow → blue → red → green →, we first randomly move selected yellow border cells (subfigure (b)), then blue border cells (subfigure (c)), then red border cells (subfigure (d)), and finally green border cells (subfigure (e)), to complete a cycle. Each of the four intermediate solutions shown in subfigures (b)–(e) is a valid solution that is considered at the EA replacement phase.

4. The PRCRX and ECMUT algorithms

In graph terminology, the problem of partitioning *n* spatial units into *k* disjoint but contiguous zones is equivalent to finding *k* disjoint connected components on the adjacency graph that cover all *n* vertices. In our SEEC implementation, both rook and queen adjacency are supported. Rook adjacency leads to a clearly planar graph while queen adjacency may not lead to a planar graph. We employ a chromosome encoding for solutions and groups with operations that rely on the adjacency graph and its associated graph operations.

4.1. Spatial crossover (PRCRX) algorithm

The PRCRX spatial crossover method has the following characteristics.

- The overlap (with its associated unique zone labeling) creates a set of connected components that mix the spatial configuration of two parent solutions. Overlapped groups serve as the basic unit for crossover recombination.
- The seeding of *k* groups defines the solution distance, which is the maximum path length. Which *k* groups should be selected as seeds is a question whose answer is closely related to the maintenance of spatial constraints and the search paths that lead to improved fitness.
- Group expansion, which is based on the selection of adjacent units in candidate sets, transforms the source solution to the target solution.
- The relinking path is a transformation of the source zone into the intermediate solutions. The mutable units chosen for expansion are determined by the fitness improvement of these intermediate solutions. Here, contiguity is enforced to avoid costly repair operations.
- The possibility of generating different and new solutions lies in the flexibility of choosing both different candidate sets as well as different units in these sets at different steps of the path.
- Randomization is invoked at many stages, including group seeding, the ordering of the candidate sets, and the selection of the adjacent candidate unit.

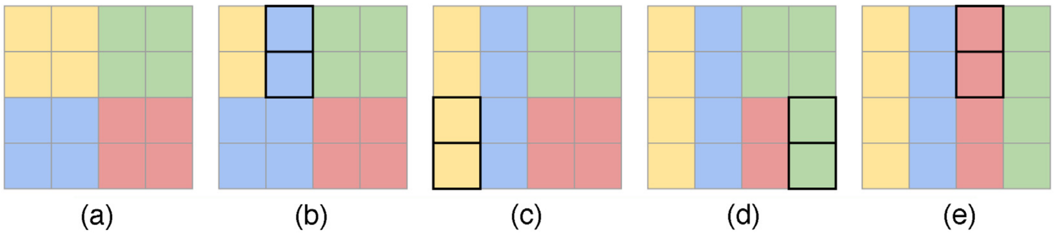


Fig. 7. Illustration of Ejection Chain Mutation (ECMUT).

Algorithm 1 A General Overlap Operation.

```
1: function OVERLAP($U, S, T$)
2:     for $u \in U$ do
3:         $i \leftarrow S[u]$
4:         $j \leftarrow T[u]$                ▷ zone assignment in the two solutions
5:         $X_{i,j} \leftarrow X_{i,j} \cup \{u\}$
6:     return $\{X_{i,j} | X_{i,j} \neq \emptyset\}$          ▷ return the overlap set
```

The path relinking process has three primary steps: overlapping, group seeding, and path building. Algorithm 1 outlines a general overlap algorithm that applies to all combinatorial chromosome encoding optimization problems. It returns the overlap, X, as a collection of group sets with a time complexity of $O(n)$. However, since a group in X is indexed using a zone label, it does not differentiate two groups with the same zone label, which is possible when two zones from S and T intersect and result in multiple connected components.

Algorithm 2 modifies the basic overlap operator to be spatially explicit. This algorithm is based on the well-known connected component labeling algorithm [58], but has been adapted to the graph representation of the problem. The algorithm applies breadth-first search (BFS) and returns the overlap, G, and its chromosome encoding, X, where $X[j]$ is the group assignment of unit j. The time complexity is $O(m + n)$, where m is the number of edges on the adjacency graph. For planar graphs, m is a constant factor of n, according to Euler's Formula. Other efficient implementations may use the Union-Find algorithm [58] within $O(n\alpha(n))$, where α is a very slow-growing inverse of the rapidly increasing Ackermann function.

The group seeding step appropriately selects k seed groups from $\{G[i], i = 1, \dots, l\}$, where l is the number of groups derived from X. The solution distance between partition S and T is $l - k$. The *overlap* algorithm creates at least k, and at most n groups. If the k seed groups are small in size such that they do not comprise many units, the solution distance will be large. In this case, the path building process is computationally costly. If the k selected seed groups are large in size, the number of available moves is limited and consequently less likely to generate desirable solutions along the path. This issue is ameliorated in large scale problems. From our empirical analysis, we find that selecting k large seed groups with unique zone labeling, while as effective as random group selection, is more efficient because the solution distance is shorter. Finally, the seeding step returns seed groups, G_t, and their corresponding candidate sets, C_t, for $t = 1, \dots, k$.

Once seed groups have been determined, we begin the path building process. Algorithm 3 presents a general path relinking algorithm based on the overlap of two solutions. It applies to non-spatial problems through the expansion of the G_t groups and the associated update to the source solution. New solutions are generated by changing a randomly selected unit in C_t to the assigned value of G_t variables. Without modification, this algorithm applies to set partitioning problems. With modifications to how C_t is constructed, the algorithm can be applied to assignment problems without the partition requirement. In this way, the coupling of overlapping and path relinking (or ejection chain) provides a new EA crossover operator.

Notice that our algorithm expands the G_t groups unit by unit, not group by group. Since the unit is the finest level of granularity, adapting to expand an arbitrary number of mutable units in C_t for a single movement is straightforward. There is a trade off. Expanding a group (i.e., a set of units) is more efficient because the number of solution evaluation steps (at line 16) is reduced. However, unit level expansion permits greater exploration of the search space.

Algorithm 4 adapts the general path relinking algorithm to a spatial path relinking algorithm. In Algorithm 3, while the FLIP() function may produce disconnected components, in Algorithm 4, this issue is bypassed by building paths based on the spatial relationships represented on the adjacency graph and graph search methods. It conducts seed group expansion on neighboring candidate units and preserves contiguity by maintaining a second adjacency graph for the zone boundary units.

As illustrated in Fig. 6, at the time of seed group expansion, we must ensure that the addition of a mutable unit does not disconnect the source zone. One way to check zone contiguity is to count, starting from a randomly picked unit, the number of connected units in the zone. If the count before the mutable unit removal is not equal to one more than the count after the removal, the zone is broken. This is an easy, though computationally expensive check that requires $O(d \times \frac{n}{k} \times \mu)$ time for k zones, where μ is the average degree of unit connectivity. Although μ is not large on a planar graph, this function must be called for each mutable unit. A more efficient way to ensure contiguity is to check whether the boundary units (using queen neighborhood) are connected only before and after unit removal. This check takes $O(d \times \phi(n, k) \times \mu')$, where the function $\phi()$, which depends on the shape of the zones, estimates the number of boundary units in a zone and μ' is the average degree of connectivity among boundary units, which is much smaller than μ. Compact shapes have small $\phi()$ values.

PRCRX randomizes the order in which the T zones are visited as well as the order for adding adjacent units in candidate sets. This randomization diversifies the decision space search. The resulting path, called *path1*, can be further optimized at little cost. Algorithm 5 implements an optimization strategy by scanning *path1* in a greedy fashion. It begins with k seed groups, formed at the beginning of the relinking process in PRCRX, and checks the kG_t groups to identify the best mutable unit to add to a new path. This process requires $O(path\_length)$ time and creates another path, called *path2*, along which another best solution is found. This process allows us to identify the best choice from the two paths as the output of the path relinking process.

4.2. Spatial mutation (ECMUT) algorithm

The ECMUT spatial mutation algorithm is designed within the same path relinking framework. The chaining process is similar. The primary difference is that, in the mutation operator, the

Algorithm 2 **The Spatial Overlap Operation.**

```
function SPATIAL_OVERLAP($U, S, T$)
    $X = [0]$                                              ▷ $X[j] = 0, j = 1, \ldots, n$
    $i = 1$                                                ▷ group index
    $Q = \emptyset$
    for $u \in U$ do
        if $u$ has been assigned a group then
            continue
        $X[u] = i$                                         ▷ create a new group
        $Q.enqueue(u)$
        while $Q \neq \emptyset$ do
            $u = Q.dequeue()$                              ▷ fetch a unit from $Q$
            $N_u = \{v \mid v$ is a neighbor of $u$ on the adjacency graph$\}$
            for $v \in N_u$ do
                if $v$ has been assigned a group then
                    continue
                if $S[u] = S[v]$ and $T[u] = T[v]$ then    ▷ same zone labels and contiguous
                    $X[v] = X[u]$                          ▷ same group
                    $Q.enqueue(v)$                         ▷ recursive search to $v$'s neighbors
        $i = i + 1$
    l = i;
    build $G = \{G[i], i = 1, \ldots, l\}$ from $X$
    return $X$ and $G$
```

Algorithm 3 **A General Path Relinking Crossover Algorithm.**

```
function FLIP($S', G_t, C_t$)
    randomly select a unit $u \in C_t$
    $G_t = G_t \cup \{u\}$
    $S'[u] =$ the source zone of $G_t$
    $C_t = C_t - \{u\}$
    return $S'$

function PRCRXO($S, T$)
    $X = overlap(S, T)$
    $\{G_t\}, \{C_t\} = seedKuniq(X, k)$        ▷ build seed groups and candidate sets
    $d = n - \sum_{t=1}^{k}(|G_t|)$             ▷ solution distance
    $S_{best} = S' = S$
    for $i = 1, \ldots, d - 1$ do
        randomly select a zone $z$ in $T$
        $flip(S', G_t, C_t)$
        evaluate $S'$
        if $S'$ is better than $S_{best}$ then
            $S_{best} = S'$
    return $S_{best}$
```

initial solution and the target solution are identical. That is, the operator searches a neighborhood along a path that begins and ends at the same solution. In ECMUT, we alternate the zones to visit, and each visit identifies a number of mutable units, at least one of which is on the zone boundary. These units are then moved to a neighboring zone, which is identified from the zone adjacency graph. Algorithm 6 shows how a general ejection chain mutation algorithm can be adapted from a path relinking framework. Algorithm 7 modifies this general algorithm to incorporate spatial considerations. It extends a previous implementation in Liu et al. [19] by allowing the chain length to be flexible in the multi-mutation strategy. Possibly, ECMUT may not be able to produce cyclic chains, but this is not required by the optimization process.

5. Empirical evaluation

We evaluate our SEEC algorithm along with the performance of the PRCRX and ECMUT operators with an application to a general spatial zoning problem that has many applications, including, for example, competitive analysis in economics [59], the drawing of school district boundaries, and land use planning [60]. This zoning problem follows the general definition of the spatial partitioning problem presented in Section 3. In this application, the objective function has two components. The first minimizes the difference in weights between the k zones. The second balances the occupancy rate of two competing agents in a spatial region. The occupancy rate, i.e. the proportional agent distribution in each spatial unit, is modeled proportional to the weight distribution between two agents. Incorporating multiple components in the objective function makes the solution landscape more complex and in turns allows a more nuanced test of algorithmic performance. The fitness of each solution is a weighted measure of the agent distribution and the weight balance of the zones. The fitness improves as the weights are increasingly balanced across the set of zones and the agent distribution is close to 50% in each zone [61]. Both objectives are defined as minimization functions, where 0 is the optimal value. Note that the fitness function is a modular component and can be flexibly specified as a weighted-sum of various components in a multi-objective optimization for any application. Contiguity is the only spatial constraint considered in our evaluation.

5.1. Implementation and case study

SEEC is implemented in C++ and parallelized using the Message Passing Interface (MPI). Liu et al. [19] developed PEAR, a high-performance computing tool for the spatial partitioning of geographic units and demonstrated its utility in a specific districting modeling application. PEAR implements a mutation operator and a basic overlay+expansion crossover operator. They found that the mutation operator was effective in searching the decision space but that the performance of their particular crossover operator was limited. SEEC is developed as a general spatial partitioning solver with both the ECMUT spatial mutation and the PRCRX spatial crossover operators integrated. A random restart feature is also implemented to handle early convergence issues.

Computational experiments were conducted on the Bridges supercomputer at the Pittsburgh Supercomputing Center and the ROGER supercomputer at the National Center for Supercomputing Applications (NCSA). Each Bridges node is configured with 2 Intel Haswell (E5-2695 v3, 2.3 GHz) CPUs, 28 cores in total, and 128 GB memory. The SEEC code was compiled on Bridges using the Intel Compiler 19.3 and Intel MPI. Each ROGER node is configured with the Intel Xeon E5-2660 processor (2.6 GHz, 20cores/node) and 256 GB memory. SEEC was compiled on ROGER using GCC 4.9.2 and MPICH 3.1.4. To enable asynchronous migration in the parallel version, we utilize MPI non-blocking functions (i.e., *MPI_Isend()* and *MPI_Iproble()*)

Algorithm 4 PRCRX: A Spatial Crossover Algorithm Based on Path Relinking.

```
1: function PRCRX($S$, $T$)
2:    $N$                                              ▷ adjacency graph for all the $n$ units. $N(u)$ returns $u$'s neighbors
3:    $X = spatial\_overlap(S, T)$                     ▷ build overlap
4:    $\{G_t\}, \{C_t\} = seedKuniq(X, k)$             ▷ build seed groups and candidate sets
5:    $d = n - \sum_{t=1}^{k}(|G_t|)$                  ▷ solution distance
6:    $S_{best} = S' = S$
7:    for zone $z \in S'$ do                           ▷ establish boundary unit adjacency graph for contiguity check
8:        build adjacency graph $BG_z$ for boundary units in $z$
9:    for zone $z \in T$ do                            ▷ build initial adjacent unit set to $G_z$ as candidate units
10:       $AC_z = \emptyset$
11:       for $u \in G_z$ do                           ▷ a candidate unit neighbors $G_z$ but in $C_z$
12:           $AC_z = AC_z \cup \{v \mid v \in N(u) \wedge v \in C_z\}$
13:       $zsgt_z =$ the zone of the initial $G_z$ in $S$    ▷ the zone where mutable unit will move
14:    $path = []$
15:    $pathlen = 0$
16:    Do
17:        $zseq =$ a random sequence of size $k$ to visit each zone in $T$
18:        for $z \in zseq$ do
19:            $mu = 0$                                 ▷ the mutable unit
20:            $cu_{best} = 0$                          ▷ stores the best candidate unit in $C_z$
21:            for $cu \in AC_z$ do
22:                $zs = S[cu]$                         ▷ the zone of $cu$ in $S$, not in $S'$
23:                if $zs = zsgt_z$ then                ▷ same zone, no effect; but update $G_z, C_z, AC_z$
24:                    $mu = cu$
25:                    break
26:                else                                 ▷ peek to see if it is mutable and with fitness improvement
27:                    check $BG$ for contiguity        ▷ whether adding $cu$ to $zsgt_z$ disconnects a zone in $S'$
28:                    if contiguity is maintained then
29:                        evaluate fitness of $S'$ if $cu$ is moved
30:                        if the $cu$ move leads to a better solution then
31:                            $cu_{best} = cu$
32:            if $mu = 0$ then                         ▷ not ineffective move
33:                if $cu_{best} > 0$ then              ▷ found an effective mutable unit
34:                    $mu = cu_{best}$
35:            $AC_z = AC_z - \{mu\}$; expand $AC_z$ to include $mu$'s neighbors
36:            $G_z = G_z \cup \{mu\}$                  ▷ update $G_z$
37:            $C_z = C_z - \{mu\}$                     ▷ update $C_z$
38:            if $mu$ is effective then
39:                update $BG$                          ▷ update boundary unit adjacency graph of $S'$
40:                $S'[mu] = zsgt_z$                    ▷ update the intermediate solution
41:                $path[pathlen{+}{+}] = mu$           ▷ record the move on the path
42:                if $S'$ is better than $S_{best}$ then
43:                    $S_{best} = S'$
44:    while there was successful expansion on any $G_t$
45:    return $path$ and $S_{best}$
```

to overlap computing and communication [62]. The algorithmic performance of SEEC, including the sequential performance comparison with other heuristics and the parallel computing performance, was conducted on Bridges. PRCRX and ECMUT operator profiling was performed on ROGER.

We employed GIS data from the state of North Carolina for our analysis.[2] The study area includes 2690 spatial units, shown in Fig. 8. Spatial adjacency, i.e. the rook and queen neighborhood matrices, is derived using open source GIS libraries, PySAL (http://pysal.org) and GDAL (http://gdal.org). The number of people residing in each spatial unit is used as the weight in our objective function. For each of the experiments reported, SEEC is configured with the parameters specified in Table 1. The coefficients of the fitness specification determine the solution landscape. The initial population is randomly generated and does not significantly affect baseline solution quality. SEEC is designed to be agnostic against specific solution landscapes and baseline solution quality. The setting of the EA population size is often a tradeoff. Large EA population sizes tend to improve the effectiveness of the crossover and mutation operators, but each iteration then requires more time. On fast machines or small problem sizes, a large EA population is preferred. The PEA settings are tuned on specific computing architecture to increase the computing and communication overlapping in a parallel computing environment.

5.2. Comparison with other heuristics

We first compare the sequential SEEC EA (with the PRCRX and ECMUT operators) with five alternative heuristics that have been designed by others for the spatial partitioning problem [42]. These other heuristics include simulated annealing, greedy algorithm, tabu search, GRASP, and GRASP (contiguous), which we developed by enhancing the basic GRASP algorithm with contiguity support.[3] Each experiment was repeated 20 times using 20 processor cores simultaneously on a single dedicated computing node on the Bridges supercomputer.

All of the runs were allowed to continue until the algorithm converged, which means that that algorithm was not able to

[2] These data are publicly available from the U.S. Census Bureau (https://www.census.gov/programs-surveys/geography/geographies/reference-maps.2011.html) and the North Carolina General Assembly web site (https://www.ncleg.gov).

[3] It is worth noting that we also studied a conventional EA package based on the *rgenoud* package in *R*. For this heuristic, during the 12-hour runtime, none of the 20 runs generated any contiguous solution. Plainly, without spatial adaptation, linear recombinations easily violate the contiguity‘ constraint on large problem instances. In general, this conclusion also applies to any population-based heuristics that only employ linear methods to generate new solutions.

Algorithm 5 A Greedy Algorithm for Optimizing a Relinked Path.

```
function PATH_OPTIMIZE($S'$, $T$, $G_t$, $path1$, $path1len$)
    $S_{best} = S'' = S$
    for zone $z \in S''$ do                                    ▷ establish boundary unit adjacency graph for contiguity check
        build adjacency graph $BG_z$ for boundary units in $z$
    for $z \in T$ do
        $muindex_z$ = index of the first moved unit in $G_z$
    $path2 = []$
    $path2len = 0$
    while $path2len < path1len$ do
        $mu = 0$
        $zsmu = 0$
        for zone $z \in T$ do
            if $muindex_z = len(G_z)$ then                      ▷ group is exhausted
                continue
            $cu = G_z[muindex_z]$                               ▷ get a moved unit and re-evaluate
            $zsgt = S''[cu]$
            check $BG$: whether adding $cu$ to $zsgt_z$ disconnects a zone in $S''$
            if contiguity is maintained then
                evaluate fitness of $S''$ if $cu$ is moved
                if the $cu$ move leads to a better solution then
                    $mu = cu$
                    $zsmu = zsgt$
        if $mu = 0$ then                                        ▷ no zones have feasible moves, end of the loop
            break
        update $BG$                                             ▷ update boundary unit adjacency graph of $S''$
        $S''[mu] = zsmu$                                        ▷ update the intermediate solution
        $path2[path2len] = mu$                                  ▷ record the move on the path
        if $S''$ is better than $S_{best}$ then
            $S_{best} = S''$
        $path2len$++
        $muindex_z$++
    return $path2$ and $S_{best}$
```

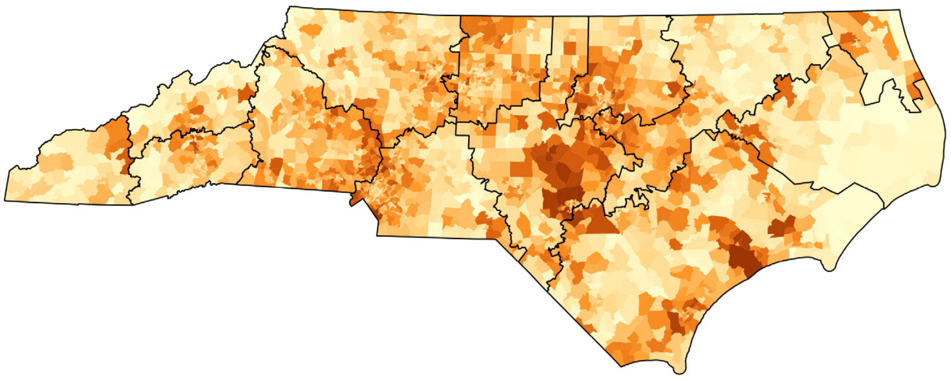


Fig. 8. Spatial units in North Carolina, colored by a weight attribute (population), overlaid with an example of a possible partition into 13 disjoint zones.

Table 1
SEEC parameter settings.

Fitness	0.2 × *(weight balance)* + 0.8 × *(agent distribution)*
PRCRX output	the better solution from PRCRX() and PATH_OPTIMIZE()
Population size	200
Selection	binary selection
Initial population	80% by region border; 20% by administration border
Elitism	on
Homogeneity check interval	20,000 iterations
Homogeneity threshold	95% population's solution distance $< 10\% \times n$
Export/import interval (PEA)	100/50
Migration rate (PEA)	2
Sending parallelism (PEA)	4

identify any further improved solutions. The best solutions identified by each algorithm are shown in Table 2. Among the five other heuristics, the GRASP (contiguous) algorithm produced the best result, reaching its best fitness, 0.0487, near the 9 h mark. Compare this performance when we utilized only the ECMUT operator. The "ECMUT snapshot" line shows when ECMUT outperformed the best result from the GRASP (contiguous) run. As we can see, ECMUT handily outperformed these other algorithms, identifying a solution with a fitness value of 0.0479 in just 56.52 s. Moreover, ECMUT continued to improve for the next 3 h, reaching its best fitness at 0.0313, as shown on the line labeled "ECMUT best". The performance of SEEC improved even more significantly when the PRCRX operator was included along with

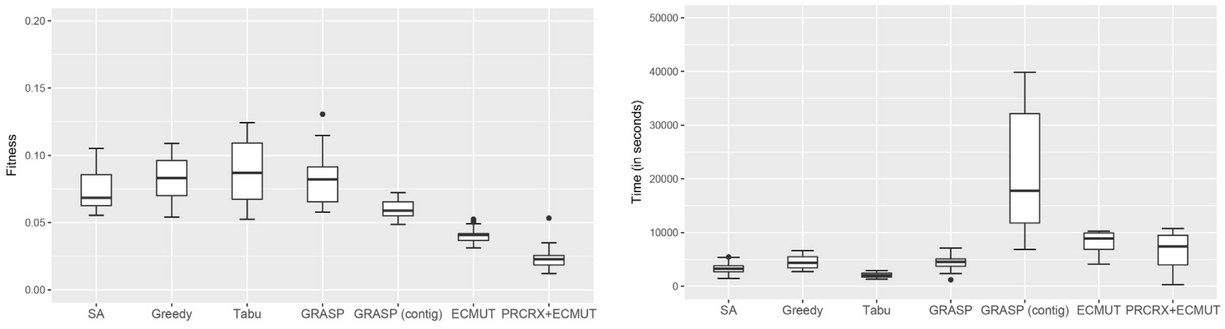


Fig. 9. Performance summary statistics.

Table 2
Performance comparison with other heuristic algorithms.

	Solution quality			Cost		
	Best fitness	Agent Distribution	Weight Balance	Time (in s)	Iterations	Iterations per second
Simulated Annealing	0.0555	0.0614	0.0317	2,922.04	728	0.25
Greedy	0.0542	0.0642	0.0143	3,020.65	108,157	35.81
Tabu	0.0525	0.0634	0.0090	2,434.10	86,838	35.68
GRASP (default)	0.0578	0.0628	0.0376	2,345.38	86,518	36.89
GRASP (contiguous)	0.0487	0.0525	0.0335	32,206.56	97,389	3.02
ECMUT snapshot	0.0479	0.0423	0.0705	56.52	2,367	41.88
ECMUT best	0.0313	0.0320	0.0286	10,120.23	1,150,203	113.65
PRCRX+ECMUT snapshot	0.0484	0.0458	0.0590	204.48	1,285	6.28
PRCRX + ECMUT best	0.0121	0.0149	0.0005	6,249.35	463,311	74.14

ECMUT, shown on the lines labeled "PRCRX+ECMUT". The snapshot line shows that PRCRX+ECMUT surpassed the best solution identified by GRASP (contiguous) in 204.48 s (or 3.4 min). The PRCRX+ECMUT run also continued to identify better solutions and finished with the fitness value of 0.0121, with notably impressive result on both agent distribution (1.49% from perfect distribution) and weight balance (0.05% away from perfect balance).

Algorithm 6 A General Ejection Chain-Based Mutation Algorithm.

```
function ECMUTO($S$, $ecLength$, $cyclic$)
    $seq$ = a random sequence of $1, \dots, n$        ▷ $n$ is the number of units
    $z_{inject} = S[seq[1]]$        ▷ zone assignment of the first ejecting unit
    for $i$: 2 .. $max(n, ecLength)$ do
        $z_{eject} = S[seq[i]]$        ▷ eject the current unit
        $S[seq[i]] = z_{in}$        ▷ unit re-assigned to the zone from the previous ejecting unit
        $z_{inject} = z_{eject}$        ▷ inject to next
    if $cyclic$ is $True$ then
        $S[seq[1]] = z_{inject}$        ▷ cyclic ejection chain
```

To measure the robustness of the performance results, summary statistics for all of the runs are computed and presented as box plots in Fig. 9. On the left, the box plots show the variation in the best fitness achieved at the convergence point where no further improvements were possible. Among the five other heuristics, the tabu solver exhibited the highest variation across 20 runs. GRASP (contiguous) had the lowest variation and produced the best median fitness among the other heuristics. However, both the ECMUT and the PRCRX+ECMUT runs exhibited better median and lower variations than GRASP (contiguous) across the set of runs. The PRCRX+ECMUT runs consistently outperformed runs with only ECMUT, though they were occasionally slower as indicated by the identified outliers. Such outliers are indicators of the difficulty and stochastic uncertainty encountered when a solver attempts to find solutions with tighter fitness bounds. Nonetheless, the SEEC algorithm consistently outperformed the five other heuristics in producing more optimal solutions.

The box plots on the right in Fig. 9 show the time it takes each heuristic to identify its best solution across multiple runs. The simulated annealing, greedy algorithm, tabu, and GRASP suffered from early convergence, which contributes to their poor performance. GRASP (contiguous) was significantly slower, but was more resilient to early convergence, and at least continued to improve, albeit slowly. In contrast, both ECMUT alone and PRCRX+ECMUT continuously identified improvements more quickly. The mean computing time for ECMUT was 8198 s, while the mean for PRCRX+ECMUT was 6561 s.

Table 2 also shows the computational cost associated with the best runs. Compared to ECMUT, the PRCRX operator is more complex and should have much higher per-iteration cost, which is not the case as reported in the last line of the table. PRCRX's rapid convergence affects the subsequent computations because the distance between solutions becomes small. Also, since most of the solutions in the population are feasible, there was no need to perform feasibility improvement in ECMUT. As a result, iterations accumulated without PRCRX and the feasibility improvement from ECMUT. The iteration speed measured at the time of surpassing GRASP (contiguous) is a more accurate gauge at 6.28 iterations per second, indicating that PRCRX, although more effective, is almost 7 times slower than ECMUT, compared to ECMUT's 41.88 iterations per second. This is not unusual since the length of the path for a crossover operation is often much longer than the length of an ejection chain for a mutation operation.

We now present results from the performance profiling of SEEC operators on the ROGER supercomputer. Since most of the plots are traces, data from a randomly chosen run for each scenario is used. We inspected all of the runs, however, to ensure that the presented results are consistent across all of the profiling runs.

5.2.1. *Performance enhancement by PRCRX*

The impact of PRCRX on the evolutionary process was examined by sampling new solutions generated in both the ECMUT and the PRCRX+ECMUT runs. The change in fitness for the feasible elite solutions is illustrated in Fig. 10. At the start of the EA, PRCRX was somewhat disruptive. PRCRX sufficiently diversified the search so that the first feasible solution occurred later than

Algorithm 7 ECMUT: A Spatial Mutation Algorithm Based on Ejection Chain.

```
1: function ECMUT($U$, $S$, $ecLength$, $blocksize$)
2:     $seq$ = a random sequence of $1, \ldots, ecLength$        ▷ randomize the order of zone visit. $ecLength$: chain length
3:     $Solutions = \emptyset$        ▷ solution set that holds improved solutions during the chaining
4:     $fitness0 = fitness(S)$
       ▷ iterative ejection chain building
5:     for $i$: $1, \ldots, ecLength$ do        ▷ initialize units in each zone
6:         $z_{eject} = (seq[i]\%k) + 1$        ▷ zone index of the eject zone; $k$ is the number of zones
7:         $Z_{eject} = Z[z_{eject}]$        ▷ unit set of the eject zone
8:         $z_{inject}$ = randomly select a receiving zone for $z_{eject}$
9:         $Z_{inject} = Z[z_{inject}]$        ▷ unit set of the inject zone
10:        $Ub_{inject} = \{ u \mid u$ is a boundary unit of zone $z_{inject} \}$
           ▷ select set of contiguous units, $\Delta$ where $\Delta \subseteq Z_{eject}$, $|\Delta| \leq blocksize$ and $\Delta$ is adjacent to $Z_{inject}$
11:        $\Delta$ = select_mutable($Z_{eject}$, $Ub_{inject}$, $blocksize$, $Z_{eject}$)
12:        if $\Delta = \emptyset$ then
13:            continue
14:        for $u \in \Delta$ do        ▷ move these units to the inject zone
15:            $S[u] = z_{inject}$
16:        if fitness($S$) is better than $fitness0$ then
17:            $Solutions = Solutions \cup \{S\}$
18:    return $Solutions$
19:
20: function SELECT_MUTABLE($P$, $B$, $maxCount$, $C$)
    ▷ $P$: a pool of contiguous units
    ▷ $B$: neighboring units to $P$, each sharing a common border to at least one unit in $P$
    ▷ $maxCount$: max number of movable units to select, as the stopping rule in the search
    ▷ $C$: an intermediate solution in which each zone's contiguity holds after MU selection
21:    $U_b = \{ u \mid u \in P$ and $u$ is adjacent to at least one unit in $B\}$        ▷ border units in $P$ to $B$
22:    $u_0$ = a unit randomly selected in $U_b$
23:    $M = u_0$        ▷ initial movable unit set
       ▷ randomized recursive traversal of the adjacency graph of $P$ to select movable units
24:    randUnitSearchRecur($P$, $M$, $maxCount$, $C$)
25:    return $M$
26:
27: function RANDUNITSEARCHRECUR($P$, $M$, $maxCount$, $C$)
    ▷ $M$: reference to the modifiable set of movable units
28:    if $maxCount = 0$ then
29:        return        ▷ recursion exit
       ▷ find neighboring units not yet included in the movable unit set
30:    $N$ = neighboring units of $M$ in $P$
31:    if $N = \emptyset$ then
32:        return        ▷ recursion exit
33:    $m$ = a random number in $[1..min(|N|, maxCount)]$
34:    $M'$ = up to $m$ randomly selected units from $N$, that keep $C$ contiguous
35:    $M = M \cup M'$
36:    randUnitSearchRecur($P$, $M$, $maxCount - m$)
```

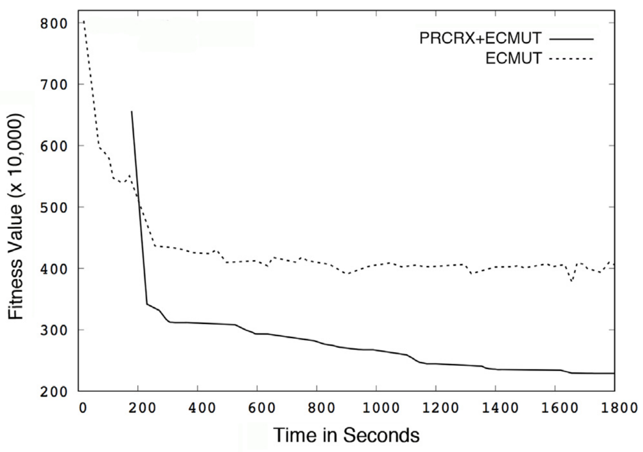


Fig. 10. Impact of PRCRX on fitness change. Lower fitness value indicates better solution quality.

it did in the ECMUT run. Shortly thereafter, however, with more feasible solutions in the population, the path relinking process became highly effective. Several apparent fitness improvements resulted from random restarts, which keep the top 10% solutions in the current population and inject the other 90% with random solutions.

5.2.2. PRCRX vs. ECMUT

To examine the difference in solution quality between PRCRX and ECMUT, we reconfigured the EA so that PRCRX and ECMUT take the same parent solutions from the population. Fig. 11 shows the solution fitness (feasible and infeasible) from the first two thousand iterations. Here, PRCRX outperformed ECMUT 91.65% of the time. In a broader sampling of 20,000 iterations, PRCRX produced a better solution quality 99.16% of the time. Two particular patterns are evident. First, fitness improved slowly with ECMUT, which is to be expected given that it embodies only small movements. Second, PRCRX produced rapid population convergence (shown in the right part of the orange curve) and was not able to progress much further until the first random restart. Hence the performance of ECMUT is comparable with PRCRX at the beginning of the search, but this pattern flips as the fitness improves. Interestingly, around the 400th iteration, there was an obvious and significant fitness improvement phase, which indicates a pivot point that is often described as the phase transition point in an optimization process. Such a transition was observed consistently in every run of the experiment.

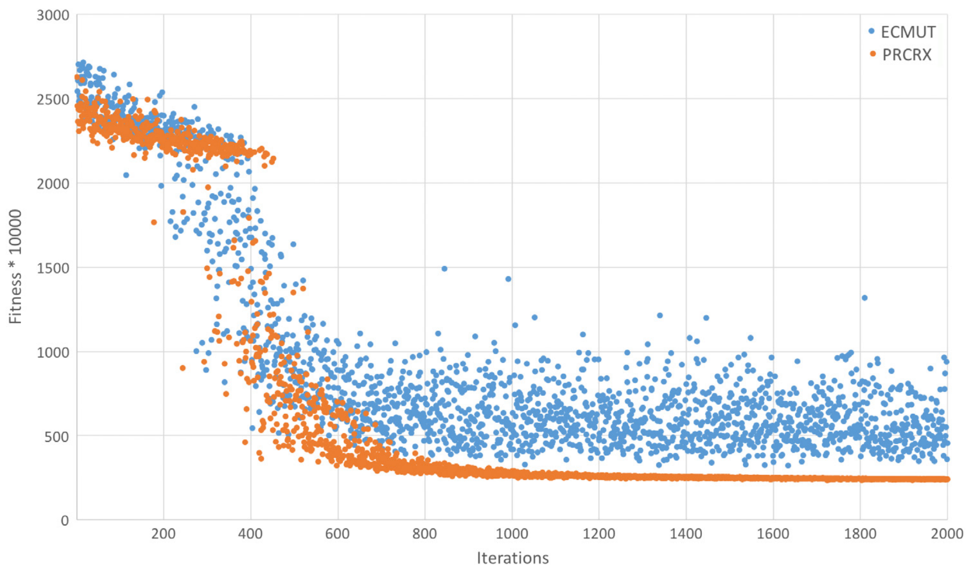


Fig. 11. Performance comparison: PRCRX vs. ECMUT in the first 2000 iterations. For minimization problems, lower fitness values indicate better solution quality.

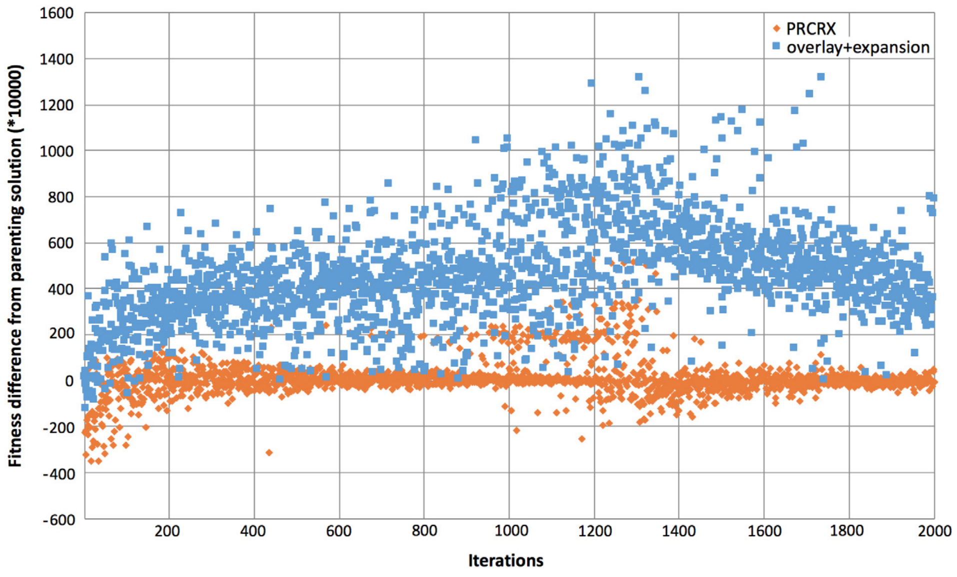


Fig. 12. Performance comparison: PRCRX and the basic overlay+expansion. Both take the same input. Fitness difference between the input and output solution is plotted. Negative difference values indicate fitness improvement.

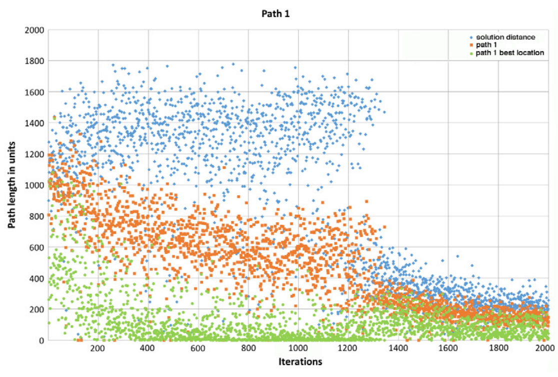
5.2.3. *PRCRX Vs. basic overlap-based recombination operators*

Overlapping is a straightforward way to combine two spatial solutions from which one generates a set of connected components of finer granularity for recombination. Based on the resulting overlap, different methods can be developed to generate new solutions. In this experiment, the PRCRX spatial crossover operator was compared with a simple overlap-based recombination method, which simply expands the k seed groups into k zones. The results are shown in Fig. 12, where the fitness change from the same parent solutions is plotted. We can see from the figure that PRCRX was able to effectively improve the fitness of the population, with the main source of fitness improvement originating from the path relinking process, while the basic overlay+expansion crossover procedure was not effective.

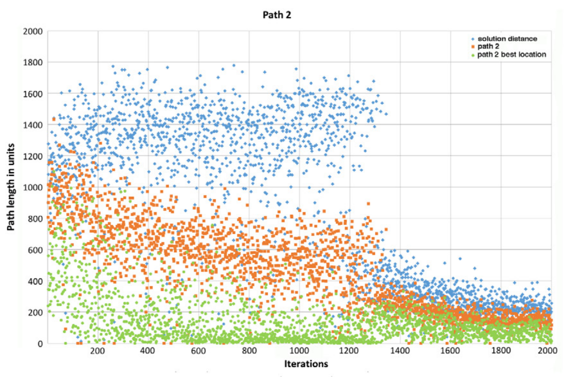
5.3. *Path analysis*

Fig. 13 plots the solution distance, path length, and the location offset of the best solution found on the path for the first 2000 PRCRX calls. Path 1, from PRCRX(), and the optimized path 2, from PATH_OPTIMIZE(), display highly similar, but not identical, patterns. First, notice the sudden drop of solution distance and path length around iteration 1200. This may indicate a pivot point at which the population became more homogeneous. It may also be the turning point from a diversified search to an intensified search. Second, the gap between solution distance and path length existed throughout the test runs, which indicates that spatial constraints may be inhibiting the transformation of the source solution, S, to the target solution, T. Third, in PRCRX, since the better of the two parent solutions is chosen as the target solution, T, one might intuit that the best solution should be found closer to T. However, the location of the best solution (green points) along the path did not seem to follow a pattern, suggesting that the distribution of better solutions is more dispersed.

We also examined whether the best solution tends to emanate from path 1 or 2 before the first random restart. On average, 68.83% of the best solutions originated from path 2, which seems



(a) Path 1



(b) Path 2

Fig. 13. Path length analysis. Solution distance (blue), path 1 length (orange), and the location of the best solution on path 1 (green) are plotted for path 1 in subfigure (a) and path 2 in subfigure (b).

Table 3
Weak scalability, which measures how long it takes (in seconds) to reach multiple fitness thresholds using different number of processors in parallel runs. A measure is left empty if a threshold was not reached within an hour.

np	Fitness Thresholds							
	0.100	0.075	0.050	0.025	0.020	0.015	0.010	0.008
10	93.99	131.60	174.62	466.04	1116.78	1785.87		
20	80.51	117.78	168.39	336.03	1135.19	1747.35		
40	35.71	110.90	146.93	250.10	389.78	1175.69	2017.33	
80	39.89	93.92	143.48	224.26	362.67	869.48	1507.02	
160	17.25	82.62	129.56	220.59	323.19	721.92	1198.01	1446.12

np = number of processor cores.

intuitive because path 2 is built upon path 1. However, 31.17% of the best solutions still arose from path 1, indicating that the exploration starting from path 1 in PATH_OPTIMIZE() may be affected by spatial constraints which inhibit further walk on path 1.

5.4. PRCRX in a parallel computing environment

This section examines the performance of our algorithm in the parallel computing environment on the Bridges supercomputer where each processor evolves an independent EA but solution exchanges are introduced via asynchronous migration among EA processes running in parallel [62]. Parallel computing enables an enormous global EA population. The results from five different experiments, using 10, 20, 40, 80, and 160 processor cores, are shown in Table 3. Each computing node utilized 20 MPI processes for a total of one hour. Each experiment was repeated 10 times, and the average is reported. We conducted weak scaling tests to measure the capability of our parallel implementation to perform more effective numerical work when more computing power was enlisted. As the number of processors increased, the size of the global population increased as well. We measured the time taken to reach multiple fitness thresholds by using different numbers of processors. Overall, as more processors were employed, SEEC was able to reach tighter fitness thresholds. Within the same tight thresholds, utilizing additional processors was associated with reduced computing time. Compared to the results from the sequential runs in Section 5.2, the parallel runs using 160 processors found solutions beyond fitness threshold 0.010 much faster. The best run achieved a fitness value of 0.0031 (with agent distribution at 0.0037, and weight balance at 0.0007), a significant improvement over the best sequential run.

6. Conclusion

An effective EA for spatial optimization must leverage information from the underlying spatial configuration to guide the solution search while adhering to the fundamental principles underlying crossover and mutation operators. Our approach preserves the features of EA operators while extending their reach with a novel spatially explicit evolutionary computation approach. The classic crossover operator is replaced with PRCRX, a spatially cognizant path relinking operator. The classic mutation operator is replaced with ECMUT, an ejection chain heuristic operator, which embeds guided spatial moves within the underlying non-linear decision space. This novel EA approach preserves contiguity in the decision space traversal and results in effective large-scale spatial partitioning.

The performance gain of PRCRX and ECMUT over the basic overlap+expansion technique is substantial. While overlapping combines two solutions, it is unclear how the intersection of the shapes at different locations preserves the desirable components of the parent solutions. Path relinking, on the other hand, leverages overlapping to obtain seeding groups, and bridges parent solutions through incrementally constructed paths. The recombination occurs along the path, allowing a controlled search for new solutions. With PRCRX and ECMUT, SEEC provides a novel EA approach with systematically designed and general spatial operators that enable effective large-scale spatial partitioning.

An additional and attractive feature, especially for very large optimization problems is that our framework seamlessly integrates into a parallel evolutionary algorithm library, allowing us to harness massive computing power that further extends the reach of the algorithm. In a parallel environment, lack of efficiency propagates and quickly degrades performance. Our spatial path relinking operator precipitously increases efficiency and provides a powerful means for the recombination of solutions identified by different processors, creating a mechanism to leverage local populations for global evolutionary computation and optimization.

SEEC provides a framework that incorporates a graph representation of the PRCRX and ECMUT designs that is flexible in its handling of other spatial elements such as adjacency, districts, and network distance and pattern. The specific local moves in the relinking paths and ejection chains can be customized for other specific applications. The geometric information of spatial units is able to incorporate both Euclidean distance as well as geometric shapes. We will extend this framework to incorporate additional spatial elements for more complex spatial configurations that may require more intensive topological and geometric processing.

Certainly, problem instances can be sufficiently large to overwhelm our innovations. There are several ways in which our work can be extended to enlarge its reach. Efficiency can be enhanced whenever early convergence can be avoided. Here, however, one must find a way to overcome the computational expense involved with detecting population homogeneity. Even the simple binary distance measure on p chromosomes requires the building of a $p \times p$ matrix and takes $O(p^2 \times n)$ to compute, where n is the number of variables. Since solution distance and path length appear to be related to population homogeneity, one might explore the effect of separating the target solution set from the population and explicitly controlling solution distance among the target solutions. In the parallel EA implementation, a distributed intensification and diversification protocol could be developed to enhance search performance by improving coordination in the evolution of the global population.

Our spatial recombination approach flexibly allows further investigation and development of effective recombination strategies for the spatial partitioning problem. The abstraction of spatial neighborhood functions and the relinking and chaining mechanisms provide the basic building blocks for developing search strategies based on the needs of specific applications. This heuristic search framework can be used to solve real-world large scale spatial optimization problems.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2020.106129.

Acknowledgments

This experiments conducted in this paper used the Extreme Science and Engineering Discovery Environment (XSEDE) resources, which are supported by National Science Foundation (NSF) grant number ACI-1548562. Specifically, it used the Bridges system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC). The performance study also used the ROGER supercomputer at the University of Illinois, which was supported by NSF Grant 1429699.

References

[1] Ningchuan Xiao, A unified conceptual framework for geographical optimization using evolutionary algorithms, Ann. Assoc. Amer. Geographers 98 (4) (2008) 795–817.
[2] Daoqin Tong, Alan T. Murray, Spatial optimization in geography, Ann. Assoc. Amer. Geographers 102 (6) (2012) 1290–1309.
[3] Johann Heinrich Von Thunen, Der isolierte staat, in: Von Thünens Isolated State, Pergamon, London, 1842.
[4] Andries M. Heyns, Jan H. van Vuuren, Multi-type, multi-zone facility location, Geogr. Anal. 50 (1) (2018) 3–31.
[5] M.E. OKelly, Locational modeling in spatial analysis: Development and maturity of concepts, in: Spatial Analysis and Location Modeling in Urban and Regional Systems, Springer, 2018, pp. 265–281.
[6] Mingjie Song, DongMei Chen, A comparison of three heuristic optimization algorithms for solving the multi-objective land allocation (MOLA) problem, Ann. GIS 24 (1) (2018) 19–31.
[7] Jing Yao, Xiaoxiang Zhang, Alan T. Murray, Spatial optimization for land-use allocation: Accounting for sustainability concerns, Int. Reg. Sci. Rev. 41 (6) (2018) 579–600.
[8] Reza Zanjirani Farahani, Nasrin Asgari, Nooshin Heidari, Mahtab Hosseininia, Mark Goh, Covering problems in facility location: A review, Comput. Ind. Eng. 62 (1) (2012) 368–407.
[9] Alan T. Murray, Maximal coverage location problem impacts, significance, and evolution, Int. Reg. Sci. Rev. 39 (1) (2016) 5–27.
[10] Alan T. Murray, Morton E. O'Kelly, Richard L. Church, Regional service coverage modeling, Comput. Oper. Res. 35 (2) (2008) 339–355.
[11] Ningchuan Xiao, Peixuan Jiang, Myung Jin Kim, Anuj Gadhave, A multi-start heuristic approach to spatial aggregation problems, in: International Conference on GIScience Short Paper Proceedings, Vol. 1, No. 1, 2016.
[12] Fahui Wang, Diansheng Guo, Sara McLafferty, Constructing geographic areas for cancer data analysis: A case study on late-stage breast cancer risk in illinois, Appl. Geogr. 35 (1) (2012) 1–11.
[13] Hongying Liu, Shuyuan Yang, Shuiping Gou, Shuai Liu, Licheng Jiao, Terrain classification based on spatial multi-attribute graph using Polarimetric SAR data, Appl. Soft Comput. 68 (2018) 24–38.
[14] Jan Faigl, Data collection path planning with spatially correlated measurements using growing self-organizing array, Appl. Soft Comput. 75 (2019) 130–147.
[15] Lavika Goel, Daya Gupta, V.K. Panchal, Hybrid bio-inspired techniques for land cover feature extraction: A remote sensing perspective, Appl. Soft Comput. 12 (2) (2012) 832–849.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., New York, 1979.
[17] Wenwen Li, Kai Cao, Richard L. Church, Cyberinfrastructure, GIS, and spatial optimization: opportunities and challenges, Int. J. Geogr. Inf. Sci. 30 (3) (2016) 427–431.
[18] Juan C. Duque, Richard L. Church, Richard S. Middleton, The p-regions problem, Geogr. Anal. 43 (1) (2011) 104–126.
[19] Yan Y. Liu, Wendy K. Tam Cho, Shaowen Wang, PEAR: A massively parallel evolutionary computation approach for political redistricting optimization and analysis, Swarm Evol. Comput. 30 (2016) 78–92.
[20] John G. Hof, Michael Bevers, Spatial Optimization for Managed Ecosystems, Columbia University Press, New York, 1998.
[21] Morton E. O'Kelly, Harvey J. Miller, The hub network design problem: A review and synthesis, J. Transp. Geography 2 (1) (1994) 31–40.
[22] John H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, USA, 1992.
[23] Sewall Wright, The roles of mutation, inbreeding, crossbreeding and selection in evolution, in: Proc. 6th Int. Cong. Genet., Vol. 1, 1932, pp. 356–366.
[24] David Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Professional, Reading, MA, 1989.
[25] C.M. Hosage, M.F. Goodchild, Discrete space location-allocation solutions from genetic algorithms, Ann. Oper. Res. 6 (2) (1986) 35–46.
[26] Yan Y. Liu, Wendy K. Tam Cho, Shaowen Wang, A scalable computational approach to political redistricting optimization, in: Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled By Enhanced Cyberinfrastructure, XSEDE '15, ACM, New York, NY, USA, 2015, pp. 1–2.
[27] Pablo Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program, C3P Report 826, 1989.
[28] Christian Blum, Jakob Puchinger, Günther R Raidl, Andrea Roli, Hybrid metaheuristics in combinatorial optimization: A survey, Appl. Soft Comput. 11 (6) (2011) 4135–4151.
[29] Devendra K. Chaturvedi, Soft Computing: Techniques and its Applications in Electrical Engineering, Springer-Verlag, Berlin, Germany, 2008.
[30] Khin Lwin, Rong Qu, Graham Kendall, A learning-guided multi-objective evolutionary algorithm for constrained portfolio optimization, Appl. Soft Comput. 24 (2014) 757–772.
[31] Omer Berat Sezer, Ahmet Murat Ozbayoglu, Algorithmic financial trading with deep convolutional neural networks: time series to image conversion approach, Appl. Soft Comput. 70 (2018) 525–538.
[32] Tomoaki Kashiwao, Koichi Nakayama, Shin Ando, Kenji Ikeda, Moonyong Lee, Alireza Bahadori, A neural network-based local rainfall prediction system using meteorological data on the internet: A case study using data from the Japan meterological agency, Appl. Soft Comput. 56 (2017) 317–330.
[33] Yu-Feng Chen, Zhan Gao, Hong Zhou, Yan Wang, Tao Zhang, Kai Che, Zheng-Tao Xiang, Traffice flow guidance algorithm in intelligent transportation systems considering the effect of non-floating vehicle, Soft Comput. 23 (19) (2019) 9097–9110.
[34] Irina Perfilieva, Javier Montero, Salvatore Sessa, Editorial to image processing with soft computing techniques, Soft Comput. 23 (6) (2019) 1777–1778.
[35] Elon Santos Correa, Maria Teresinha A Steiner, Alex A Freitas, Celso Carnieri, A genetic algorithm for solving a capacitated p-median problem, Numer. Algorithms 35 (2–4) (2004) 373–388.
[36] Jonas Schwaab, Kalyanmoy Deb, Erik Goodman, Sven Lautenbach, Maarten J. van Strien, Adrienne Grêt-Regamey, Improving the performance of genetic algorithms for land-use allocation problems, Int. J. Geogr. Inf. Sci. 32 (5) (2018) 907–930.
[37] Nozomi Hitomi, Daniel Selva, Incorporating expert knowledge into evolutionary algorithms with operators and constraints to design satellite systems, Appl. Soft Comput. 66 (2018) 330–345.
[38] S. Openshaw, A geographical solution to scale and aggregation problems in region-building, partitioning and spatial modelling, Trans. Inst. Br. Geographers (1977) 459–472.
[39] S. Openshaw, L. Rao, Algorithms for reengineering 1991 census geography, Environ. Plan. A 27 (3) (1995) 425–446.

[40] Daoqin Tong, Alan Murray, Ningchuan Xiao, Heuristics in spatial analysis: a genetic algorithm for coverage maximization, Ann. Assoc. Amer. Geographers 99 (4) (2009) 698–711.
[41] Ningchuan Xiao, David A. Bennett, Marc P. Armstrong, Using evolutionary algorithms to generate alternatives for multiobjective site-search problems, Environ. Plan. A 34 (4) (2002) 639–656.
[42] Roger Bivand, Geocomputation and open source software: components and software stacks, NHH Dept. of Economics Discussion Paper (23), 2011.
[43] S.D. Minor, T.L. Jacobs, Optimal land allocation for solid and hazadous waste landfill siting, J. Environ. Eng. 120 (1994) 1095–1108.
[44] C.J. Brookes, A parameterized region-growing programme for site allocation on raster suitability maps, Int. J. Geogr. Inf. Sci. 11 (1997) 375–396.
[45] Thomas J. Cova, Richard L. Church, Contiguity constraints for single-region site search problems, Geogr. Anal. 32 (4) (2000) 306–329.
[46] Justin C. Williams, A zero-one programming model for contiguous land acquisition, Geogr. Anal. 34 (4) (2002) 330–349.
[47] Takeshi Shirabe, A model of contiguity for spatial unit allocation, Geogr. Anal. 37 (1) (2005) 2–16.
[48] Takeshi Shirabe, Districting modeling with exact contiguity constraints, Environ. Plan. B: Plann. Des. 36 (6) (2009) 1053–1066.
[49] Alan T. Murray, Tony H. Grubesic, Ran Wei, Spatially significant cluster detection, Spat. Stat. 10 (Suppl. C) (2014) 103–116.
[50] Thomas P. Runarsson, Xin Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Trans. Evol. Comput. 4 (3) (2000) 284–294.
[51] Hesam Izakian, Witold Pedrycz, A new PSO-optimized geometry of spatial and spatio-temporal scan statistics for disease outbreak detection, Swarm Evol. Comput. 4 (2012) 1–11.
[52] Douglas M. King, Sheldon H. Jacobson, Edward C. Sewell, Wendy K. Tam Cho, Geo-graphs: An efficient model for enforcing contiguity and hole constraints in planar graph partitioning, Oper. Res. 60 (5) (2012) 1213–1228.
[53] James MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1 Statistics, University of California Press, Berkeley, California, 1967, pp. 281–297.
[54] Fred Glover, Genetic algorithms and scatter search: Unsuspected potentials, Stat. Comput. 4 (2) (1994) 131–140.
[55] Fred Glover, Manuel Laguna, Rafael Marti, Fundamentals of scatter search and path relinking, Control Cybernet. 29 (3) (2000) 653–684.
[56] Mutsunori Yagiura, Toshihide Ibaraki, Fred Glover, An ejection chain approach for the generalized assignment problem, INFORMS J. Comput. 16 (2) (2004) 133–151.
[57] Mutsunori Yagiura, Toshihide Ibaraki, Fred Glover, A path relinking approach with ejection chains for the generalized assignment problem, European J. Oper. Res. 127 (2) (2006) 548–569.
[58] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, third ed., The MIT Press, 2009.
[59] Gaia Nicosia, Andrea Pacifici, Ulrich Pferschy, Competitive subset selection with two agents, Discrete Appl. Math. 159 (16) (2011) 1865–1877.
[60] Paul N. Courant, On the effect of fiscal zoning on land and housing values, J. Urban Econom. 3 (1) (1976) 88–94.
[61] Hamid Hamoudi, Marta Risueno, The effects of zoning in spatial competition, J. Reg. Sci. 52 (2) (2012) 361–374.
[62] Yan Y. Liu, Shaowen Wang, A scalable parallel genetic algorithm for the generalized assignment problem, Parallel Comput. 46 (2015) 98–119.