Pergamon

Computers & Geosciences Vol. 22, No. 8, pp. 837–847, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0098-3004/96 $15.00 + 0.00

PII: S0098-3004(96)00022-2

A RASTER-BASED C PROGRAM FOR SITING A LANDFILL WITH OPTIMAL COMPACTNESS

JEHNG-JUNG KAO

Institute of Environmental Engineering, National Chiao Tung University Hsinchu Taiwan, China
(*e-mail*: jjkao@green.ev.nctu.edu.tw)

(Received 27 October 1995; accepted 15 February 1996)

Abstract—Landfill siting requires performing spatial analyses for various factors to evaluate site suitability. A geographical information system, although capable of effectively manipulating spatial data, lacks the capability to locate an optimal site when compactness and other factors are considered simultaneously. In our previous work, a mixed-integer compactness model was proposed to overcome this difficulty. However, computational time with a conventional mixed-integer programming package for solving the model is time consuming and impractical. Therefore, in this work, a C program is developed, based on a proposed raster-based branch-and-bound algorithm. The program can implement multi-factor analyses for compactness and other siting factors with weights prespecified by the user. An example is provided to demonstrate the effectiveness of the program. Copyright © 1996 Elsevier Science Ltd

Key Words: Compactness, Spatial analysis, Landfill siting, Geographic Information System, Optimization, Raster, C.

INTRODUCTION

Numerous factors must be considered when siting a landfill. Zyma (1990) suggested that an appropriate landfill have minimum impact on environment, society, and economy, comply with regulations, and receive general public acceptance. The decision maker might reach either an inappropriate decision without thoroughly reviewing all prevailing regulations and factors or not grasp fully the background information of a candidate site. In addition to the overall condition, spatial data should be collected for these factors to assess related impacts. A landfill siting analysis generally requires extensive effort to evaluate the considered factors. Implementing such a complicated procedure in a conventional information processing approach would be expensive and tedious. A Geographic Information System (GIS) is capable of processing a large amount of spatial data, thereby saving time that would be spent normally in selecting an appropriate site. Lindquist (1991) stated that using a GIS for landfill site selection increases objectivity and flexibility. Relatively easy presentations of GIS siting results are among its advantages also.

The GIS utilized herein is GRASS (USACERL, 1993), a publicly accessible GIS. GRASS is a raster-based GIS operated on a UNIX platform, although limited vector-type functions are also available. In the raster mode, spatial data are divided into cellular geo-referenced objects. Factors involving the features of a geographical object are expressed with numbers and linked to the GIS cell that represents the object. A collection of connected GIS cells is termed a map layer; each map layer stores a feature of an area. The primary advantage of applying such a raster-based GIS is the simplicity of its data storage and processing, thereby making it easy to combine with other tools.

Although a GIS is useful in siting applications, the algorithm for obtaining the optimal site, with simultaneous consideration of site compactness and other factors, generally is unavailable. Compactness represents the nature of the site and the extent to which it can be regarded as integrated tightly. The lower the level of compactness, the less likely the solution is to satisfy siting requirements, subsequently making general land planning difficult. On the other hand, the more compact the selected site is

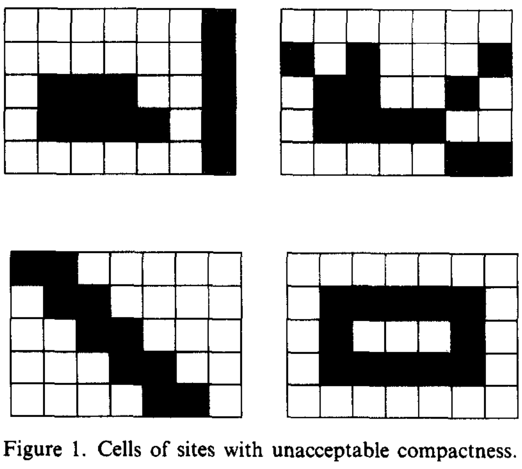
Figure 1. Cells of sites with unacceptable compactness.

the closer together the land, thereby making the site easier to manage. Selected land cells that are not contiguous are not suitable for a landfill site. Moreover, the selected cells that are contiguous but have a poor shape are also inappropriate. Figure 1 presents several unacceptable situations. Therefore, a compactness model must be applied to ensure the appropriate shape of the selected site.

Compactness can be defined by a variety of methods (Wright, ReVelle, and Cohon, 1983; Gilbert, Holmes, and Rosenthal, 1985; Diamond and Wright, 1989). For instance, Wright, ReVelle, and Cohon (1983) used the ratio of the perimeter to the area of a site as a measure of compactness. According to this definition, the shorter the perimeter of a site implies the higher its degree of compactness. Diamond and Wright (1989) applied the ratio of the largest diameter square to the area of a selected site as another measure. The largest diameter refers to the longest distance between any two points within the selected site. However, this method of calculation is nonlinear. Minor and Jacobs (1994) and Benabdallah and Wright (1992) adopted the former definition for a waste landfill siting problem and a land allocation problem, respectively. These spatially compact models, although useful in solving a siting problem, have not been integrated into a raster-based GIS. Diamond and Wright (1989) indicated that such an integration would provide an intelligent decision-making tool for land-use problems. The major obstacle to this integration is that significant numbers of integer variables and constraints are required to construct a compactness model for raster-based GIS map layers, thereby making the model difficult to solve by a general mixed-integer programming package. In our previous work (Kao and Lin, 1995), we proposed an improved compactness model for raster-based GIS data. In that work, the model was applied to a case study in central Taiwan using XMP/Zoom (Marsten, 1988), a mixed-integer programming package. The package, although capable of solving the model, requires too much computing time for a raster-based landfill siting problem and therefore is impractical. Primary reasons for this computational problem include (1) the large number of necessary steps to search for a feasible integer solution during the branch-and-bound solution procedure implemented by XMP/Zoom, and (2) the unnecessary branches on cells that are not contiguous. Once a set of cells is selected, the corresponding feasible integer solution can be determined easily without using a complex programming method. Moreover, branching on obviously impossible cells that are far away from previously selected cells is unnecessary. The XMP/Zoom branch-and-bound procedure, although effective for many mixed-integer programming problems, is not efficient for the raster-based siting problem. Therefore, in this study, a branch-and-bound algorithm is developed specifically for the siting problem to improve the

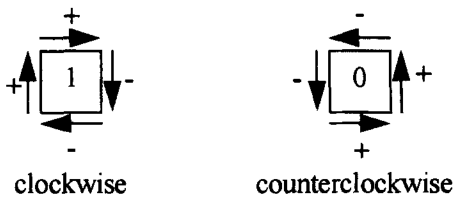

Figure 2. Directed cell sides.

computational efficiency. A Perl (Wall and Schwartz, 1991) program was written to implement the developed branch-and-bound algorithm and compactness model. However, the computational time was still too long, although faster than XMP/Zoom. The program is therefore rewritten in C language to accelerate the computation. Firstly in this paper, the developed compactness model is outlined, the structure and main functions of the C program are then described, and finally, a sample problem is presented to demonstrate the application of the program.

COMPACTNESS MODEL

Perimeter calculation

In this work, as used by Wright, ReVelle, and Cohon (1983), the compactness of a site is defined by the ratio of its perimeter to its area. This study focuses primarily on raster-based data. A site is comprised of adjacent square GIS cells uniform length of side. Based on this framework, $I_{i,j}$, an [0,1] indicator variable, is defined to represent whether cell i, j belongs to a considered site. When the value of the variable is 1, the cell is a part of the site; if the value is 0, the cell is not a part of the site. Each cell-side length is directed positively or negatively according to the value of $I_{i,j}$. Figure 2 illustrates how the directed sides are defined. When a cell is part of the site, $I_{i,j}$ equal to 1, the directions of cell sides are defined clockwise. On the other hand, if a cell is not part of the site, $I_{i,j}$ equal to 0, then its cell side directions are defined counterclockwise. For computational convenience, each side length is designated to be 0.5 units, and the directed length pointed toward south or west is negative. With this concept of directed sides, each side length of a cell can be defined by the following equations:

$$\begin{aligned} LT_{i,j} &= -0.5 + I_{i,j} \\ LL_{i,j} &= -0.5 + I_{i,j} \\ LR_{i,j} &= 0.5 - I_{i,j} \\ LB_{i,j} &= 0.5 - I_{i,j} \end{aligned} \tag{1}$$

where $LT_{i,j}$, $LL_{i,j}$, $LR_{i,j}$, and $LB_{i,j}$ are the top, left, right, and bottom side lengths of cell i, j, respectively.

Any side of a cell must be adjacent to one side of another cell. According to Figure 2, when the values of $I_{i,j}$ for any two adjacent cells are the same, that is, both equal to 1 or 0, the sum of the directed lengths of the common side of both cells will be zero, and therefore the side is not part of the perimeter. On the other hand, where the values of $I_{i,j}$ for two adjacent cells are different, the sum of the two directed lengths of the common side will be $+1$ or -1, and therefore the side is part of the perimeter. In calculating the perimeter of a site, the valid perimeter of a cell side is the sum of its directed side lengths of two adjacent cells that share the same side. The valid perimeters of the top and right side of a cell, i, j, can be computed by the following formula:

$$\textit{Top side}: SLT_{i,j} = LT_{i,j} + LB_{i,j-1}$$

$$\textit{Left side}: SLL_{i,j} = LL_{i,j} + LR_{i-1,j} \qquad (2)$$

When the $I_{i,j}$ of the cell is 1, $LT_{i,j}$ and $LL_{i,j}$ are 0.5. At the same time, $LB_{i,j-1}$ and $LR_{i-1,j}$ may be 0.5 (positive) or -0.5 (negative) depending on whether the neighboring cell is part of the site or not. Thus, the value of $SLT_{i,j}$ and $SLL_{i,j}$ must be either 1 or 0. On the other hand, when $I_{i,j}$ has a value of 0, $SLT_{i,j}$ or $SLL_{i,j}$ must be either 0 or -1. Because the vectors of the segments on a closed curve should sum to zero, the total length of the of "top" plus "left" valid perimeters of a site should be equivalent to the total length of "bottom" plus "right" ones; in addition, both total lengths have opposite positive and negative values. As such, the site perimeter can be determined by merely calculating the "top" and "left" valid perimeter lengths. The valid perimeter of a cell, SL, can therefore be defined by the following equation:

$$SL_{i,j} = SLT_{i,j} + SLL_{i,j} = 2I_{i,j} - I_{i,j-1} - I_{i-1,j} \quad (3)$$

Possible values of $SL_{i,j}$ are 0, 1, -1, 2, and -2. When the value is other than 0, the absolute value of this value expresses the number of sides of the cell that contribute part of the perimeter of a candidate site. A linear programming model cannot directly calculate the absolute value; thus, a new nonnegative variable, $V_{i,j}$, is introduced. This yields the following constraint:

$$2I_{i,j} - I_{i,j-1} - I_{i-1,j} + V_{i,j} \geq 0 \text{ for all } i, j \qquad (4)$$

When $SL_{i,j}$ is less than 0, $V_{i,j}$ will be equal to its absolute value. When $SL_{i,j}$ is larger than 0, $V_{i,j}$ is equal to 0. Thus, the total of all $V_{i,j}$ values represents the sum of all negative (or positive) $SL_{i,j}$ values. As mentioned for vectorial balance of a closed curve, the sum of all positive values should be equal to the sum of all negative values. The total of all $V_{i,j}$ values is equal therefore to half of the site perimeter. Figure 3 shows a sample site of twelve cells. As shown in this figure, the above calculation indicates that the sum of all negative side lengths is equal to the sum of all positive side lengths for cells with sides on the boundary of the site.

The model

According to the perimeter calculation described above, the spatial compactness model proposed in this study is summarized as follows:

$$\text{Min} \sum_{i=0}^{i=m} \sum_{j=1}^{j=n+1} V_{i,j}$$

subject to

$$2I_{i,j} - I_{i-1,j} - I_{i-1,j} + V_{i,j}$$
$$\geq 0 \ \forall i \in \{0,\ldots,m\};\ \forall j \in \{1,\ldots,n+1\}$$

$$\sum_{i=1}^{i=m} \sum_{j=1}^{j=n} I_{i,j} \geq A \ \forall i \in \{1,\ldots,m\};\ \forall j \in \{1,\ldots,n\}$$

$$\sum_{i=1}^{i=m} \sum_{j=1}^{j=n} C^k_{i,j} I_{i,j} \geq G^k \ \forall k \in \{1,\ldots,p\}$$

where m and n are the number of columns and rows of cells that represent the entire siting area; A is the required size (in numbers of cells) of the desired site; $C^k_{i,j}$ is the value of siting factor k for cell i, j; p is the number of considered factors; and G^k is the lower bound of the sum of factor values of cells in a site for siting factor k. Notably, ensuring that each cell in the siting area has an adjacent cell requires that a pseudo-column of cells (for $j = n + 1$) be added on the left-hand side of the siting area; a pseudo-row (for $i = 0$) of cells also is added on the top of the siting area. Consequently, the continuity of the selected cells of the solution to the above model is guaranteed because the model seeks the smallest perimeter.

For the proposed compactness model, only one constraint is required for each cell, whereas the Wright, ReVelle, and Cohon (1983) model requires two constraints and the Minor and Jacobs (1994) model requires eight constraints. For the number of required variables, the proposed model requires only one integer and one noninteger variable, whereas the Minor and Jacobs model requires three integer variables and the Wright model requires five integer variables. As for a mixed-integer linear programming

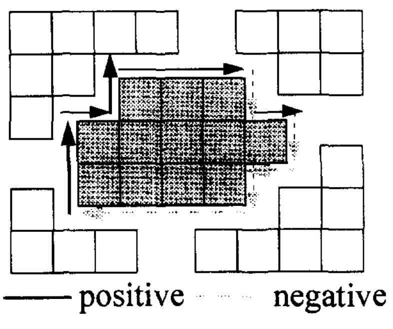

Figure 3. Directed perimeter.

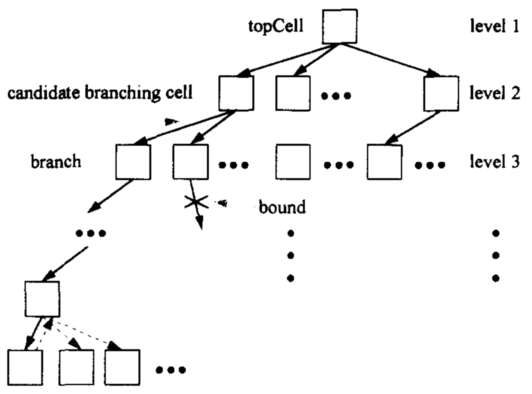

Figure 4. Traversing tree for implementing proposed depth-first branch-and-bound algorithm.

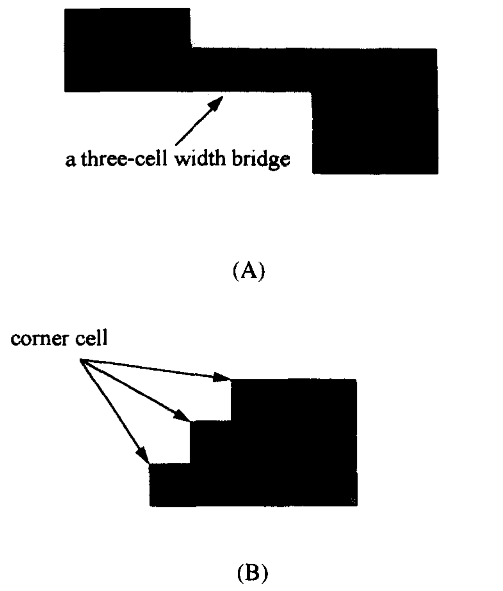

Figure 6. A, Sample site with horizontal three-cell width bridge and B, sample site with three corner cells.

model, increasing the number of integer variables increases rapidly the computational time required to solve the model. However, increasing the number of noninteger variables does not have such a significant effect. Therefore, decreasing the required number of integer variables of the proposed model is useful particularly in reducing the computing time.

Model with multiple factors

For application to a problem with multiple factors, the objective function of the formulation (Eq. 5) should be modified as follows:

$$\text{Min} \sum_{i=0}^{i=m} \sum_{j=1}^{j=n+1} \left(w_v V_{i,j} + \sum_{k=1}^{p} w_k C^k_{i,j}\right) \tag{6}$$

where w_v is the weight for compactness and w_k is the weight for factor k. This model is formulated on the basis of the weighting method described by Cohon (1978).

BRANCH-AND-BOUND ALGORITHM

The siting model described is a mixed-integer optimization model. Optimization packages such as XMP/Zoom (Marsten, 1988) generally use a branch-and-bound algorithm to solve this type of model. However, the general branch-and-bound algorithm cannot solve the raster-based siting problem effectively because many inappropriate cells are branched. Therefore, a depth-first branch-and-bound algorithm is proposed to improve the computational efficiency when searching for a site with optimal compactness.

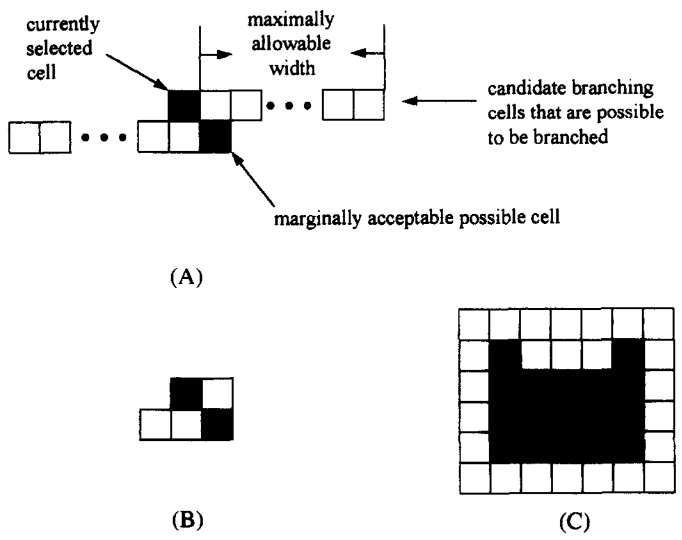
currently selected cell
maximally allowable width
candidate branching cells that are possible to be branched
marginally acceptable possible cell
(A)
(B)
(C)

Figure 5. Cells that can be branched, A, possible cells; B, previous branching approach; and C, a sample case that can be missed with the previous branching approach.

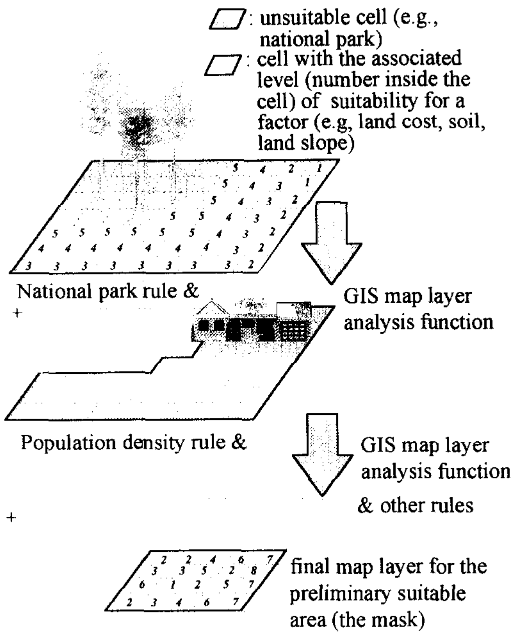
: unsuitable cell (e.g., national park)
: cell with the associated level (number inside the cell) of suitability for a factor (e.g, land cost, soil, land slope)
National park rule &
+
GIS map layer analysis function
Population density rule &
GIS map layer analysis function
& other rules
+
final map layer for the preliminary suitable area (the mask)

Figure 7. Typical procedure of applying GIS analysis functions to obtain mask and factor suitability map layers.

The two major steps of *Branch* and *Bound* of the algorithm are described next.

Branch. The algorithm starts with selecting a top cell into the cell set of a possible site, as indicated by the topCell shown in the branch-and-bound tree illustrated in Figure 4. Next, all candidate branching cells for the topCell are added into a branching pool. A candidate branching cell is one that is not eliminated after applying bounding rules described later for the *Bound* step of this algorithm. One of the candidate branching cells is then selected (branched), down to the next level of the tree; in each level a cell is selected. The candidate branching cells for this newly added cell are collected as the branching pool of the new cell. This procedure is repeated until the number of selected cells satisfies the required siting size. Such a set of selected cells is termed a site and is then checked for its feasibility and noninferiority. If the site passes the check, objective and factor values of the site are recorded, and the best values obtained so far are used as bounds. After a site is checked, another site is formed by replacing the most recently selected cell by another cell in the same branching pool. Again, the new site is checked, and, if passed, its objective and factor values are recorded once more. These values, if superior to all previous values, are recorded as the best bounds. This procedure is repeated until no new cell in the branching pool in the current level can be selected. The searching process then moves up one level in the tree (see Fig. 4) and continues the checking procedure from another candidate branching cell in the branching pool of the upper level. If no new cell can be selected in the branching pool of the upper level, the searching process moves further up in the tree. This procedure is repeated until all branching pools are empty and each cell has been used as a topCell. This branching process is basically a depth-first searching procedure to traverse cells on the tree.

Bound. The size of the branching tree can be increased rapidly with an increase in the number of marked cells available in the siting area, thereby making the problem difficult to solve within an acceptable time. Therefore this step is applied to prune subtrees of the branching tree that are not necessary to explore. Pruning the subtrees as early as possible saves a significant amount of computing time. Two sets of bounding rules are provided in this function to prune subtrees.

The first set includes four rules that are always checked, that is: possible cells, bounds of factor values, bounds of estimated site factor values, and improvement of the objective function value based on an estimated current objective value. Figure 5A illustrates possible cells to be branched next for the currently selected cell. Because all cells are traversed in row order and from left to right, branching backward to those cells having been traversed is unnecessary. Furthermore, the required size of a site is known before applying the program, thereby making it unnecessary to branch on cells that are too far away from the currently selected cell. In an earlier version of the program, branching was performed only for cells adjacent to the currently selected cell, as illustrated in Figure 5B. However, such a branching approach misses good solutions sometimes such as the one in Figure 5C. Although the branched cells are not immediately adjacent to the currently selected cell, contiguity of the finally selected site is guaranteed because a site with unconnected cells has a poor compactness value and will not be selected.

The estimated value of a factor for a site is computed by the following equation:

$$F_e = \sum_{i=1}^{s} f_i + \sum_{j=s+1}^{r} f_j' \tag{7}$$

where F_e is the estimated value of a factor for the currently selected site; s is the number of currently selected cells; f_i is the factor value of cell i; r is the number of cells required for a site; and f_j' is the lower bound of the factor value of cell j. The value of compactness can be determined easily by checking the two cells on the left and top sides of each selected cell. According to Equation (4), if any left or top cell is not selected, the compactness value is increased by 1. The estimated objective function value can then be determined by the estimated site factor values and the compactness value. Moreover, comparing the estimated objective function value against the best one recorded so far reveals that the subtree following the

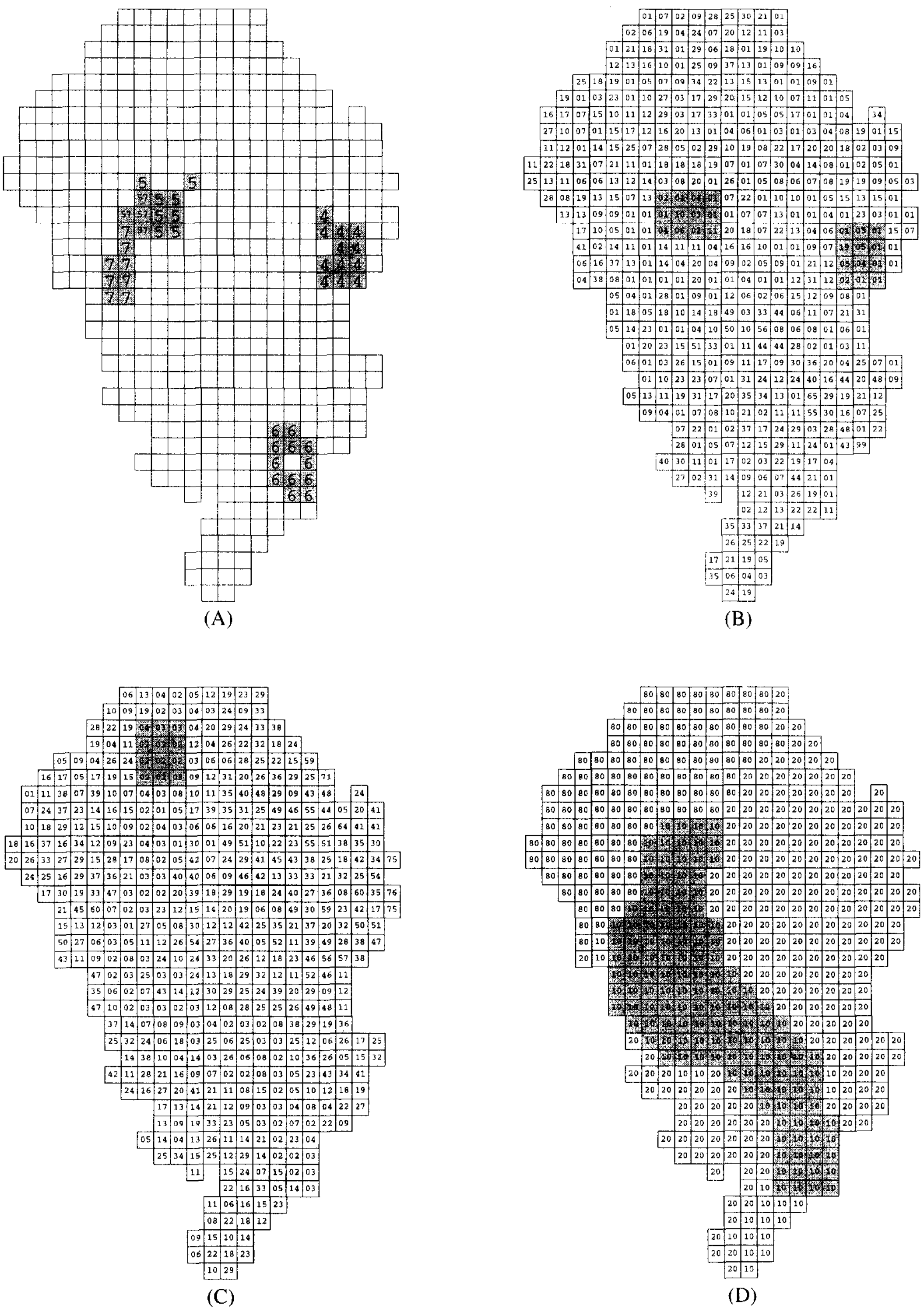

Figure 8. Mask and factor map layers with solutions for sample problem: A, mask and solutions for cases 4–7; B, factor map layer for land cost with solutions for case 1; C, factor map layer for land slope with optimal solution for case 2; and D, factor map layer for soil with optimal solutions for case 3.

Table 1. Listing of pseudocode

```
main():
    read mask and factor map layers;
    read user-provided options with readUserProvided();
    for each marked cell, topCell, (rowwise and left to right) do loopbranch();
    report the result.
loopBranch(): for ease of explanation bounding rules are not listed in the same order shown in
      the program.  Instead they are grouped into two categories of non-optional and optional
      rules.)
    implement bounding rules:
          -possible cells only;
          -bounds of factor values;
          -bounds on estimated site factor values;
          -objective function improvement based on an estimated objective function value;
    implement selected optional bounding rules:
          -maximally allowable width and/or height to topCell;
          -maximally allowable width of a horizontal bridge;
          -maximally allowable number of corner cells;
    if a possible site is found, do validCheck() for checking feasibility and/or noninferiority;
    collect candidate branching cells that can be branched into array adjacentCell;
    for each cell in adjacentCell do loopBranch();
validCheck():
    check feasibility with constraints provided by checkConstraint();
    check noninferiority by noninferior():
    if it is valid site, output the site information.
```

current cell can be pruned if the estimated one is worse than the current best one.

The other set of bounding rules include: maximally allowable width and height to the topCell, maximally allowable width of a horizontal bridge, and maximally allowable number of corner cells. Options are provided for the user to adjust the default values of these rules. Maximally allowable width and height to the topCell are set to avoid branching on those cells that are too far away from previously selected cells. Maximally allowable width of a horizontal bridge is set to avoid searching for sites having a poor shape. As shown in Figure 6A, a bridge exists between two land parcels. The default rule is set to avoid any bridge of a selected site. A corner cell is defined as a selected cell without any cell that is selected on its left-hand and top sides, as illustrated in Figure 6B. If the number of such corner cells is limited to 1, all sites having rectangular shapes led by the corner cell are evaluated. For a new problem, defined limits are recommended to solve the problem of finding the initial solution. Subsequently, the limits can be relaxed with the objective function value of the first solution used as a bound to avoid searching solutions inferior to the first one.

Properly using these rules can reduce significantly the size of the branch-and-bound tree. With this special branch-and-bound algorithm, the computing time is significantly less than that used by XMP/Zoom. Additional rules can be added to this function to improve further the computational speed. However, a rule should not be added if too many steps are required to implement it. A complex rule may prune several subtrees; however, the computational time spent in each branch for checking the rule may not justify the time saved from subtrees it prunes. For instance, although the vertical bridge width can be checked by adding additional statements into the program, this check is not as simple as checking the horizontal bridge width because cells are traversed horizontally from left to right instead of vertically.

Table 2. Tested weight sets for sample problem

Factor	Compactness	Land cost	Land slope	Soil
Case 1	100	1	0	0
Case 2	100	0	1	0
Case 3	100	0	0	1
Case 4	10	10	10	0
Case 5	10	10	0	10
Case 6	10	0	10	10
Case 7	50	10	10	10

Cells that are not eliminated in this *Bound* step are termed "candidate branching cells". If any candidate branching cell exists in the branch-and-bound tree, the *Branch* step is repeated by branching out of one of the candidate branching cells. The two steps are repeated until no candidate branching cell is available on the tree.

THE C PROGRAM

The structure

The pseudocode listed in Table 1 describes the structure of the C program developed to implement the raster-based compactness model with the branch-and-bound algorithm.

Before applying the program, the user must prepare a map layer to specify the siting area, where a site is to be located. This map layer is termed the mask map layer. The value of each cell of this mask map layer is either 1 or 0, indicating that the cell is either part, or not part, of the siting area. Such a map layer can be created by GRASS from an existing map layer after excluding obviously unsuitable areas. Appendix 1 presents a sample list of GIS cell values for the mask used in the example described later. In addition to the mask, at least one factor map layer should be provided to evaluate the suitability of a site. For the objective function, different weights can be assigned for a range of factors and the compactness.

Primary C functions of the program are described next in the order of **loopBranch**, **validCheck**, and **readUserProvied**, followed by a description of how to use the program.

Function **loopBranch**: *branch-and-bound algorithm*

Function **loopBranch** is a recursive function. It is the most important function in this program for executing the depth-first branch-and-bound algorithm specifically designed for solving the raster-based siting problem.

Function **validCheck**

Two functions of **checkConstraint** and **noninferior** are called by this function to check a possible site. User provided constraints are specified in **checkConstraint** and, if asked, **noninferior** is initiated to check the noninferiority of a site. Both functions are optional to the user. If the user attempts to provide additional constraints in **checkConstraint**, the program must be recompiled to reflect the change. A noninferior site is defined as a site for which no other possible sites are strictly superior to it for both factor and objective function values.

Function **readUserProvided**

This function allows the user to define options from an ASCII file without recompiling the entire program. The file format and related options are described in the next sub-section for the usage of the program.

Usage

The program can be executed with the following command:

```
sitecomp < mask_file> <factor_file_1>

<factor_file_2>...
```

where `sitecomp` is the program name, `<mask_file>` is the file name of the mask map layer, and `<factor_file_n>` is the file name of the map layer for factor *n*. At least one `<factor_file_n>` must be provided. Appendix 1 provides a sample of the format of mask and factor files. For a mask file, all values are either 1 or 0, and for a factor file any value can be presented. The size of factor files should be the same as that of the mask file in numbers of columns and rows. One pseudo-column, must be added to the left-hand side of the map, and one pseudorow is added to the top of the map layer to guarantee that each marked cell has cells immediately adjacent to it on the left and top sides for computing compactness values.

An additional file, whose name is defined by the constant `USERPROVIDEDF` in the program, also must be provided to set appropriate options for the problem to be solved. Appendix 2 presents a sample of this file. The following options are available for the program:

1. Minimal and maximal sizes of the site to be searched (option names: **AREAmin** and **AREAmax**);
2. Weights in the objective function, see Equation (6) (option names: `w0`, `w1`,..., etc.);

3. Upper and lower bounds of values for each factor (option names: `u1`, `u2`,..., and `l1`, `l2`,..., etc.);

4. Upper and lower bounds of summation of values for cells in a site for each factor (option names: `U1`, `U2`,..., and `L1`, `L2`,..., etc.);

5. The best objective value currently known, or a best estimate (option name: `currBestObj`);

6. The optimal direction of the objective function (option name: `objOptDir`);

7. The optimal direction of each factor (option names: `o1`, `o2`,..., etc.);

8. An option to check noninferiority of a site (option name: `checkNoninferior`);

9. Maximally allowable horizontal bridge width (option name: `checkHcon`);

10. Maximally allowable width and height to the topCell (option names: `maxWidthTo1st` and `maxHeightTo1st`);

11. A limit on the allowable number of corner cells (option name: `maxCornerCells`); and

12. An option to avoid overlapping of selected sites (option name: `alternative`).

SAMPLE PROBLEM

A sample data set from Shihu County in central Taiwan is presented to demonstrate the effectiveness of the program. A mask map layer and the suitability values of three factors of land cost, land slope, and soil for the mask area are prepared. The mask map layer is created by GIS map analysis functions provided by GRASS to exclude obviously inappropriate areas. Figure 7 shows a typical procedure for creating such a map layer. The factor suitability map layers are created by a scoring system based on their values. For instance, high land cost is assigned a low suitability value, and vice-versa. In this sample problem, to formulate the objective function easily, a low value implies a high suitability and a high value implies a low suitability.

Figure 8 illustrates the mask map and associated factor maps. The number of marked cells in the mask map layer is 518. Appendix 2 lists one of the option files used for this problem. The C program is applied to identify the optimal site for each factor, with weights for other factors in the objective function being set to zero. The weight for compactness is set to be significantly larger than the weight for each factor to guarantee the solution with optimal compactness. The associated factor map of Figure 8 displays the solution. This figure reveals that with a different factor, a different solution may be obtained. Moreover, the optimal solution may not be unique. For instance, for the cost factor example, two alternative optimal solutions with the same objective value are obtained. For the soil factor, many alternative optimal solutions are possible.

The problem is tested further with two or all three factors (other than the compactness) considered simultaneously. Table 2 summarizes the weight sets tested. Figure 8A illustrates solutions for examples 4–7. Because the weights assigned to the compactness are insufficiently large to dominate the accumulated effect of other factors, the solutions obtained for these four examples are not so compact as the solutions for examples 1–3. The larger the weight of the compactness implies the greater the compactness of the obtained solution.

CONCLUSION

The C program developed in this study is capable of locating the optimal site within a candidate area and simultaneously considering site compactness and other factors. The program is primarily used for data created from a raster-based GIS. The branch-and-bound algorithm proposed specifically for a raster-based siting problem effectively reduces the size of the searching tree if options provided to implement the algorithm are selected properly. The computing time required for solving a siting problem is reduced significantly. Multiple factors with prespecified weights can be included also. Appropriate factors and assigned weights should, however, be carefully evaluated. Different sets of factors and weights may yield different solutions, as indicated by the solutions in Figure 8 for the sample problem. Determining an appropriate set of siting factors with appropriate weights requires evaluating the relative importance and associated utility functions of factors by close interaction with decision makers. Decision makers may assign a different set of factors with different weights after they have evaluated solutions similar to those in Figure 8. The program may then be applied iteratively several times until the decision makers accept the final solution. The source code is available by anonymous FTP from the server IAMG.ORG.

Acknowledgments—The author would like to thank many friends on the Internet network who helped with the use of GRASS in this work and the Construction Engineering Research Laboratory, Army Corps of Engineers, Champaign, Illinois for providing the publicly accessible GRASS. Appreciation to Mr James Darrell McCauley of Purdue University for his assistance in establishing an interface program for data exchange between the developed program and GRASS. Also, Mr Hung-Yue Lin is appreciated for his assistance in preparing data for the sample application and valuable discussion regarding some branch-and-bound rules.

REFERENCES

Benabdallah, S., and Wright, J. R., 1992, Multiple subregion allocation models: Jour. Urban Planning and Development, v. 118, no. 1, p. 24–40.

Cohon, J. L., 1978, Multiobjective programming and planning: Academic Press, New York, 50 p.

Diamond, J. T., and Wright, J. R., 1989, Efficient land allocation: Jour. Urban Planning and Development, v. 115, no. 2, p. 81–96.

Gilbert, K. C., Holmes, D. D., and Rosenthal, R. E., 1985, A multiobjective discrete optimization model for land allocation: Management Science, v. 31, no. 12, p. 1509–1522.

Kao, J.-J., and Lin, H. Y., 1995, Geographic information system for municipal solid waste landfill siting and evaluation (II): Report to Miaoli Prefecture, Taiwan, 144 p.

Lindquist, R. C., 1991, Illinois cleans up: using GIS for landfill siting: Geographic Information Systems, February 1991, p. 30–35.

Marsten, R., 1988, XMP user's guide: Dept. Management Information Systems, Univ. Arizona at Tucson, Tucson, Arizona, 50 p.

Minor, S. D., and Jacobs, T. L., 1994, Optimal land allocation for solid- and hazardous-waste landfill siting: Jour. Environmental Engineering, v. 120, no. 5, p. 1095–1108.

USACERL, 1993, GRASS 4.1 user's reference manual: U.S. Army Construction Engineering Research Laboratory, Champaign, Illinois, 555 p.

Wall, L., and Schwartz, R. L., 1991, Programming Perl: O'Reilly and Associates, Inc., variously paged.

Wright, J., ReVelle, C., and Cohon, J., 1983, A multiobjective integer programming model for the land acquisition problem: Regional Science and Urban Economics, v. 13, no. 1, p. 31–53.

Zyma, R., 1990, Siting considerations for resource recovery facilities: Public Works, September 1990, p. 84–86.

APPENDIX 1

Sample Mask Map Layer

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

APPENDIX 2

Sample Option File

```
# This is a sample file for $USERPROVIDED.  (see sitecomp.c for file name.)
# Lines with a leading '#' are comment lines. Blanks are ignored.
# Set approriate values of options will reduce the size of the B&B tree.
# Format for each option is
#  optionName [=] value [;]   # although '=' and ';' can be omitted, it
#   is better to keep them for better readability

# Must define AREAmin and AREAmax
         AREAmin=12;
         AREAmax=12;

# wx: for objWeight [x]; x=0 is for compactness value
         w0 = 100; # for compactness
         w1 = 1; # for factor 1
         w2 = 0; # for factor 2
         w3 = 0; # for factor 3

# lx, ux: for facLB[x] and FacUB [x];NOTE: facLB [0] is not used.
#    actually, LB and UB will be computed, so if you set
#       LB < actual LB, I will charge it to LB = actual LB; and
#       UB > actual UB, I will charge it to UB = actual UB.
# e.g, l1=1; or u1=10;

# Lx, Ux: for siteFacLB[x], siteFacUB [x]; NOTE; siteFacLB [0] is  meaningless
# siteFacUB [0] is for compactness value. Note  U0 is not checked in
# ValidCheck(). So, you may still see some compact values > U0.
#       U0 = 10;

# Define current known best ojective value. (or a best guess)
#       currBestObj = 800;

# If there are several alternative sites and do not allow overlapping.
# You may use this option to disable overlapping.
#        alternative = false;

#objOptDir=tominimize; to define tominimize or tomaximize for MIN or MAX

#ox=tominimize; # for optimization direction for each factor

#checkNoninferior default: true; please see description provided in the paper.
checkNoninferior=false;

#checkBestObj default: false; if false, will report all processed solutions.
#checkBestObj=true;

#checkHcon default: 0; # of continuous horizontal grids of a bridge
checkHcon=3;

#maxWidthTo1st default=AREAmax; actual width should add 1
maxWidthTo1st=6;

#maxWidthTo1st default=AREAmax:
maxHeightTo1st=6;

#maxCornerGrids default=AREAmax/2.0:
maxCornerGrids=3;
```