A Bilevel Model for a Tower Defense Game

F. Liberatore[a,*], D. Camacho[b]

[a] *Department of Computer Science, Viterbi School of Engineering, University of Southern California, Los Angeles, CA.*
[b] *AIDA, Computer Science Department, Autonomous University of Madrid, Madrid, Spain.*

Abstract

In the last years, the videogame industry has grown considerably. One of the possible research topics in videogames is related to the design of intelligent agents, capable of efficiently solving games or providing an interesting challenge to players. A game solver can also be used to calculate evaluation measures in an automatic content generation framework, as well as to assess the skill of the players. Operations Research methodologies and techniques can easily find application in this field. In this paper we consider a Tower Defense game played on a grid where the player can locate towers to prevent enemies to reach a goal. We propose a bilevel model for the maximization of the enemies' exposition to fire. Two different solution methodologies are proposed: a single level reformulation for the bilevel model and a decomposition algorithm. Both approaches are empirically tested and their performances are evaluated. We show that, while the single level reformulation is extremely inefficient, the decomposition algorithm is very effective.

Keywords: Tower Defense, Interdiction Models, Bilevel Programming, Decomposition Algorithm, Videogames.

1. Introduction

Creating programs capable of playing grand-master level chess was the dream of early pioneers of computing and information theory such as Alan Turing and Claude Shannon, that even described in a general way how this might be achieved. Much of the academic work on games has focused on traditional board games, such as chess and checkers, obtaining many successes in the struggle to beat expert human players. Lately, computer games have increased exponentially their quality, their diversity, and their pervasiveness. This improvement has been recognized and acknowledged by a growing audience, so much that in 2012 the worldwide computer games market was estimated to be worth around USD 78bn annually, and is predicted to continue to grow rapidly (Nayak and Baker, 2012). Traditionally, games have long been seen as an ideal test-bed for the study of artificial intelligence (AI). On the other hand, the Operations Research (OR) community has dedicated little attention to them, although many commercial computer games make use of very well-known OR techniques. Given their characteristics (e.g., frequently changing real-time environment, uncertain inputs from the user, limited computational resources), computer games provide a challenging framework to test the capabilities of OR techniques, albeit not that different from real-world decision-making. The wide range of problem-solving techniques and methods encompassed by OR easily finds application in the field of games and, as we show in this paper, can be very successful. The OR community possesses the capabilities

*Corresponding author.
Email addresses: fliberat@usc.edu (F. Liberatore), david.camacho@uam.es (D. Camacho)

and tools to become one of the main points of reference in videogames, a rich and mostly unexplored field full of new and exciting research opportunities.

In this work, we tackle a game of Tower Defense (TD). The game is played on a grid. A group of enemies originates at a starting cell and moves to a goal cell following the shortest path. For each enemy that reaches the goal, the player loses one life. To prevent it, the player can spend resources to locate towers that attack the enemies within their range. An enemy cannot traverse a cell occupied by a tower and the player cannot locate the tower so as to completely obstruct the enemies. Different types of towers are considered, having different attributes. We propose a model for the definition of the tower configuration (i.e., towers' locations and types) that maximizes the enemies' exposition to fire in a TD game played on a generic graph. Specifically, the problem is formulated as a bilevel model. Given their nested structure bilevel models cannot be solved by means of standard Mixed-Integer Programming (MIP) methods (Colson et al., 2007). Therefore, we propose two solution approaches: the first one is a single-level reformulation using Karush–Kuhn–Tucker (KKT) optimality conditions (Karush, 1939; Kuhn and Tucker, 1951), and the second one is an ad-hoc decomposition algorithm. We test both solution methodologies empirically to compare their efficiency in terms of time taken to solve a problem to optimality. Also, as in commercial videogames the computational power assigned to the decision engine is generally very limited, we compared the quality of the solutions found when the time limit is set to five seconds. The most obvious application for the proposed model is the development of an intelligent agent capable of solving a TD game. Obtaining optimal (or good) tower configurations is the first step toward the development of advanced gamers behaviors analyses, such as players modeling, strategy identification, or error detection. Also, a model that identifies the best tower configuration for a level can be used to guide a software for automatic contents generation (Browne and Maire, 2010), such as the creation of new levels or of new game elements, like tower types and enemies. Finally, giving its idiosyncrasies, the proposed bilevel model finds application in the military field; e.g., to plan for protection against unmanned drones.

As explained in Section 2, the model presented in this paper innovates the state-of-the-art on TD controllers in a number of ways. First, this is the first work in the field to explicitly applying bilevel optimization and OR methodologies. Second, all the previous contributions considered only one type of tower and no upgrades are possible, while our model considers multiple types of towers. Tower upgrades can be included as tower types, by adjusting their cost to include those of the preceding types. Finally, to the best of the authors knowledge, this is the first optimization model that can be applied to a real-time TD-based computer game.

The paper is structured as follows. In the next section, a review of the research in the area is presented. In Section 3 the proposed model and solution methodologies are explained. The results of the experiments are illustrated in Section 6. Finally, some conclusions and suggestions for possible research opportunities are given in Section 7.

2. Background

TD games are a genre of strategy games focusing on resource allocation and unit placement. In it's simplest form, a TD game consists of a human player buying and organizing defensive towers that fire upon a set deployment of different types of offensive enemies[1]. For each enemy destroyed by the towers, the player earns resources corresponding to the difficulty of the enemy. If enough of the enemies are destroyed (usually all the enemies that appear in a wave), the player wins the round. If however enough enemies

[1] In the literature on TD games, the enemies are often referred to as "critters."

Table 1: Controllers for Tower Defense: References and structural characteristics.

Contribution	Tower Types	Methodology	Tower Upgrades	# Enemy Types
Huo et al. (2009)	1	PSO and GAs	No	1
Rummell (2011)	1	Adaptive ad-hoc algorithm	No	1
Tan et al. (2013)	1	GAs + NNs	No	1

reach the end of the player's map, the player loses. The tower deployment usually involves buying, placing and upgrading towers that fire automatically on the forces. In recent years TD games have attracted the attention of researchers in the field of Computer Science. Avery et al. (2011) defined the TD game for the purpose of future research. More specifically, the authors provided a brief history of the TD genre, described the structure of TD games, and identified research areas. According to the authors, the list of potential fields of interest includes automatic map generation, intelligent agents for playing the game, enemies strategy, automatic game element generation, dynamic difficulty adjustment, and player modeling. However, all the contributions in the field dealt with the definition of intelligent controllers. Huo et al. (2009) compared Particle Swarm Optimization (PSO) and Genetic Algorithms (GAs) for the identification of optimal tower locations on a map. Both algorithms performed similarly in the experiments, showing a convergence time greater than 80 seconds even for simple maps. Therefore, the authors stated that these techniques cannot be applied to a real-time context and suggested combining them with Neural Networks (NNs). A few years later, Tan et al. (2013) combined GAs and NNs to evolve intelligent controllers to determine tower locations. Two types of neural networks were tested, namely, Feed Forward NNs and Elman Recurrent NNs. The experimental results showed that the former performed better than the latter. The authors applied the resulting controllers to the evaluation of map designs, with the aim of reducing the time and cost involved in designing map by automatizing certain testing procedures (e.g., difficulty level classification and identification of unsolvable maps). Again, the computational time required to evolve the NNs are not reported in the paper. A different approach is investigated by Rummell (2011) that proposed an adaptive ad-hoc algorithm that ranks candidate locations based on multiple runs of the game and assigns towers according to this ranking. Although the algorithm seemed to perform fairly well, unfortunately no information regarding the computational time required for its training was reported in the paper. Table 1 summarizes the main characteristics of the contributions on controllers for TD. Thanks to the table it can be easily seen that the number of works in this field is still very limited and that research focused more on exploring the applicability of different techniques to very simplified TD scenarios.

The model proposed in this paper presents several improvements. First, this is the first work in the field to explore the applicability of bilevel optimization and OR methodologies. Second, the model has been formulated to consider any number of tower types. Tower upgrades can be included in the optimization process by translating them into additional tower types with adjusted costs, i.e., their cost must include those of the preceding types. Finally, as illustrated in Section 6, this is the first model that identifies good tower configurations with limited computational resources and, therefore, can be successfully applied to a real-time computer game.

From the perspective of optimization, TD games can be seen as a special kind of Interdiction Model (IM). IMs focus on the identification of the most critical components in a system. This family of models, first introduced by Wollmer (1964), has been extensively studied over the past few decades, especially within the context of network problems. The analysis of network interdiction models has been performed with respect to different reliability measures, such as connectivity, distance (or cost) and capacity. A multitude of early network interdiction models are surveyed by Church et al. (2004), while a survey of the latest contributions can be found in Losada et al. (2009). Most of the researches in this field focused on the following underlying models:

- Network Flow (Granata et al., 2013; Matisziw and Murray, 2009). The objective of Network Flow Interdiction models is the identification of the components (e.g., arcs and edges) that optimally disrupt

network operations (e.g., minimizing the maximum source-sink flow or maximally disconnecting the network) when removed.

- Shortest Path (Israeli and Wood, 2002). The objective of Shortest Path Interdiction models is to determine the elements (generally, arcs and edges) that would increase the shortest path length the most if removed.
- Location (Aksen et al., 2014). The objective of Location Interdiction models is, generally, to define the set of most critical elements (e.g., nodes, vertices, arcs, and edges) that increases the cost of the underlyinhsystem the most when removed.

IMs are normally modeled and solved as bilevel programs. Bilevel optimization was first realized in the field of game theory by the von Stackelberg (2011) that described a hierarchical problem consisting of a *leader* and a *follower*. In a Stackelberg game, the players of the game compete with each other, such that the leader makes the first move, and then the follower reacts optimally to the leader action. Both the leader and the follower have complete knowledge of the problem and the reciprocal strategies. Therefore, to optimize its objective, the leader needs to take into consideration the optimal response of the follower. In this setting, the leader optimization problem (also called the upper-level or the leader-level) contains a nested optimization task that corresponds to the follower's optimization problem (also called the lower-level or the follower-level). This nested structure is very typical in bilevel programs. Unfortunately, because of this very structure, bilevel models cannot be solve directly by means of standard MIP methods (Colson et al., 2007). In the following, the main optimal solution approaches are briefly presented.

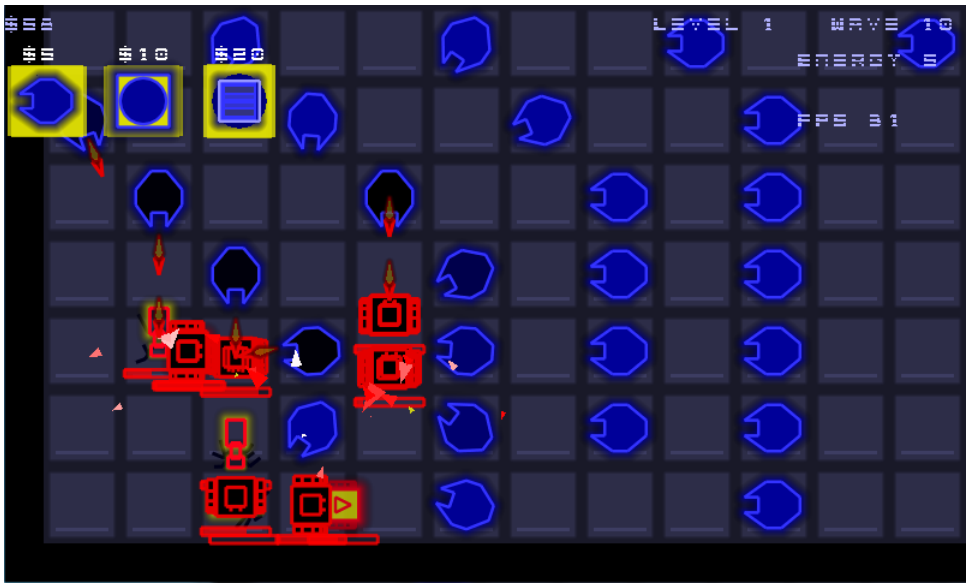
1. Enforce the KKT optimality conditions (Karush, 1939; Kuhn and Tucker, 1951) for the follower level, then linearize the resulting complementary slackness restrictions, and finally solve the resulting single-level program by means of a standard MIP solver. This methodology tends to be inefficient due to the linearization step that generally results in a high number of binary variables.
2. Dualize the follower level, then linearize the non-linear terms in the objective function, and finally solve the resulting single-level program by means of a standard MIP solver. This methodology can be applied only when the leader and the follower have the same objective function.
3. Bender's decomposition method.
4. Ad-hoc methods, e.g., implicit enumeration algorithms (Scaparra and Church, 2008a,b) and decomposition algorithms (Israeli and Wood, 2002).

In the next section, the problem is formalized and a bilevel formulation for TD games is presented.

3. Tower Defense Game Problem

The problem under analysis regards a TD game played on a grid (Figure 1 shows a sample TD game having the structure considered in this work). The player has assigned a fixed budget that she can spend to locate towers on the cells of the grid. Different tower types are considered, each characterized by a cost, a range (e.g., maximum distance of the affected cells), and a fire power (e.g., the amount of damage dealt to enemies traversing affected cells). A group of enemies appears on a source cell and move toward a sink cell following the shortest path on the grid. Only one type of enemy is considered. The objective of the game is to defend the exit from the advancing enemies. Maximizing the damage delivered to the enemies is the best way to ensure that they do not reach the exit. Thus, we now present a model that maximizes the total fire exposure for the enemies, called MXFE. Although we apply the MXFE in a TD game played on a grid, we formulate and solve the model for any type of graph.

Figure 1: Screenshot of the open source TD game (Kamphausen and Giersch, 2012) having the same structure as the TD model addressed in this article. Enemies are represented in red and towers are drawn in different shades of blue. In the top-left corner the game displays the resources available and the tower types with the corresponding prices.



3.1. Sets

$G = (N, E)$, Complete graph.

N, Set of nodes, indexed by i and j. Nodes represent potential tower locations.

E, Set of edges, indexed by e. e^s and e^t identify the starting and the termination node for edge e.

$FS(i)$, Forward star for node i, i.e. set of edges directed out of node i.

$RS(i)$, Reverse star for node i, i.e. set of edges directed into node i.

$s, t \in N$, Source and sink nodes, respectively.

K, Set of types of tower, indexed by k.

3.2. Parameters

R, Resource available.

d_e, Length of edge e.

c_k, Cost of locating a tower of type k.

a_{ie}, Interdiction matrix. 1 if locating a tower at node i interdicts edge e (i.e., when $i = e^s \vee i = e^t$); 0 otherwise.

f_{ikj}, Fire intensity matrix. Amount of fire power received by node j when a tower of type k is located at node i.

3.3. Variables

x_{ik}, Location variables. 1 if a tower of type k is located at node i; 0 otherwise.

y_e, Path variables. 1 if edge e belongs to the shortest $s-t$ path; 0 otherwise.

z_j, Node traversing variables. 1 if node j is traversed by path represented by variables $\mathbf{y}$; 0 otherwise.

3.4. Constraints

Bi-Level Maximum Fire Exposure model (BL-MXFE):

$$\max \quad \sum_{i\in N}\sum_{k\in K}\sum_{j\in N} f_{ikj}x_{ik}z_j \tag{1}$$

$$s.t. \quad \sum_{i\in N}\sum_{k\in K} c_k x_{ik} \leq R \tag{2}$$

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

$$z_i \leq \sum_{e\in E|e^s=i\vee e^t=i} y_e \qquad \forall i \in N \tag{4}$$

$$0 \leq z_i \leq 1 \qquad \forall i \in N \tag{5}$$

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

$$\mathbf{y} \in \arg\min \sum_{e\in E} d_e y_e \tag{7}$$

$$s.t. \quad \sum_{e\in RS(i)} y_e - \sum_{e\in FS(i)} y_e = \begin{cases} -1 & \text{if } i=s, \\ 1 & \text{if } i=t, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i \in N \tag{8}$$

$$y_e \leq 1 - \sum_{k\in K} x_{ik} \qquad \forall i \in N, e \in E\,|a_{ie}=1 \tag{9}$$

$$y_e \geq 0 \qquad \forall e \in E \tag{10}$$

Constraints (1)-(6) represent the leader level, that is concerned with the definition of the tower configuration. Objective function (1) maximizes the shortest path fire exposure, calculated as the sum of the fire intensity at each node traversed by the enemies' path. Budget constraint (2) enforces that the total towers' cost must be within budget. Constraints (3) state that a tower can be only of one type. Constraints 4 set the node traversing variables. Finally, constraints 5 and (6) define the node traversing variables and the tower location variables, respectively.

Constraints (7)-(10) represent the follower level, that is, an Interdicted Shortest Path model (ISP). The objective function (7) minimizes the total path length. Constraints (8) are classical path balance constraints, while constraints (9)are the arc interdiction constraints, i.e., an edge cannot be traversed if a tower is located either at its starting node or its ending node. Path variables are defined in constraints (10).

Given its nested structure, BL-MXFE cannot be solved in this form by means of standard MIP methods. Therefore we propose a single level formulation and a decomposition algorithm, illustrated in the Sections 4 and 5, respectively.

4. Single-Level Reformulation for the MXFE

We now present a single-level reformulation for the MXFE based on the KKT conditions (Karush, 1939; Kuhn and Tucker, 1951). First of all, we need to linearize the objective function (1) as it contains a non-linear term. To do so we need to introduce the following continuous variables and constraints:

$$\mu_{ikj} \leq x_{ik} \quad \forall i, j \in N, k \in K \tag{11}$$

$$\mu_{ikj} \leq z_j \quad \forall i, j \in N, k \in K \tag{12}$$

We can now convert the BL-MXFE into a single-level problem by substituting the follower level by its corresponding KKT conditions. Let α_i, β_{ie}, and γ_e be the dual variables associated to constraints (8), (9), and (10), respectively. By taking variables x_{ik} as constants we can define the KKT conditions as follows:

$$(8) - (10) \tag{13}$$

$$\alpha_{e^t} - \alpha_{e^s} + \textstyle\sum_{i \in N | a_{ie}=1} \beta_{ie} + \gamma_e - d_e = 0 \quad \forall e \in E \tag{14}$$

$$\beta_{ie} \left(y_e - 1 + \textstyle\sum_{k \in K} x_{ik} \right) = 0 \quad \forall i \in N, e \in E \,|\, a_{ie} = 1 \tag{15}$$

$$\gamma_e y_e = 0 \quad \forall e \in E \tag{16}$$

$$\beta_{ie} \leq 0 \quad \forall i \in N, e \in E \,|\, a_{ie} = 1 \tag{17}$$

$$\gamma_e \geq 0 \quad \forall e \in E \tag{18}$$

Constraints (14) are stationarity conditions. The complementary slackness conditions (i.e., (15) and (16)) are non-linear, therefore we need to linearize them by introducing the following binary variables and constraints:

$$\beta_{ie} \geq T_{ie} \tau_{ie} \quad \forall i \in N, e \in E \,|\, a_{ie} = 1 \tag{19}$$

$$y_e - 1 + \textstyle\sum_{k \in K} x_{ik} \geq \tau_{ie} - 1 \quad \forall i \in N, e \in E \,|\, a_{ie} = 1 \tag{20}$$

$$\gamma_e \leq \Upsilon_e \upsilon_e \quad \forall e \in E \tag{21}$$

$$y_e \leq 1 - \upsilon_e \quad \forall e \in E \tag{22}$$

$$\tau_{ie} \in \{0, 1\} \quad \forall i \in N, e \in E \,|\, a_{ie} = 1 \tag{23}$$

$$\upsilon_e \in \{0, 1\} \quad \forall e \in E \tag{24}$$

where the T_{ie} and the Υ_e are "big enough" constants. For a discussion regarding feasible values for T_{ie} andΥ_e, please refer to subsections "On the Value of T_{ie}" and "On the Value of Υ_e," respectively. We can now formulate the MXFE as a single-level program.

Single-Level Maximum Fire Exposure model (SL-MXFE):

$$\max \quad \sum_{i\in N}\sum_{k\in K}\sum_{j\in N} f_{ikj}\mu_{ikj} \tag{25}$$

$$s.t. \tag{26}$$

$$\text{Objective Function Linearization:} \quad (11), (12) \tag{27}$$

$$\text{Leader Level:} \quad (2), (3), (5), (6) \tag{28}$$

$$z_i \leq \sum_{e\in E|e^s=i\vee e^t=i} \overline{y}_e \qquad \forall i \in N \tag{29}$$

$$\text{Follower Level KKT Conditions:} \quad (8)-(10) \tag{30}$$

$$(14), (17), (18) \tag{31}$$

$$(19)-(24) \tag{32}$$

$$\text{Path Variables Binarization:} \quad \overline{y}_e \leq y_e \qquad \forall e \in E \tag{33}$$

$$\overline{y}_e \in \{0,1\} \qquad \forall e \in E \tag{34}$$

Binary variables $\overline{y}_e$ have been introduced as the integrality condition on the solution of the shortest path problem is not enforced by the KKT conditions. In fact, without the binary variables $\overline{y}_e$, the program would assign fractional values to the path variables y_e, so as so set to 1 as many node-traversing variables z_i as possible. Therefore, constraints 29 bind the value of variables z_i using binary variables $\overline{y}_e$. As illustrated in Section 6, this formulation is very inefficient and can only solve problems on very small graphs. For this reason in Section 5 we present a decomposition algorithm capable of tackling larger instances.

4.1. On the Value of T_{ie}

Given constraints (19), constants T_{ie} must be smaller than or equal to the value of β_{ie}. Variables β_{ie} are associated to interdiction constraints (9). Therefore, their value corresponds to the improvement in the ISP (i.e., Interdicted Shortest Path) solution value resulting from "de-interdicting" edge e by removing a tower located in node i. A good theoretical upper bound is given by the following:

$$T_{ie} \leq \text{MXSP}\left(\textstyle\sum_k x_{ik} = 0, y_e = 1\right) - \text{MXSP}\left(\textstyle\sum_k x_{ik} = 1\right) \quad \forall i \in N, e \in E \,|a_{ie} = 1 \tag{35}$$

where $\text{MXSP}(\cdot)$ is the objective function value of the MXSP obtained where the condition between brackets $(\cdot)$ is applied. Calculating the left-hand-side of this equation requires solving two separate MXSP, which would be impossible. Therefore, we consider the following valid bound:

$$T_{ie} \leq \underline{\text{MXSP}}\left(\textstyle\sum_k x_{ik} = 0, y_e = 1\right) - \overline{\text{MXSP}}\left(\textstyle\sum_k x_{ik} = 1\right) \quad \forall i \in N, e \in E \,|a_{ie} = 1 \tag{36}$$

where $\underline{\text{MXSP}}(\cdot)$ and $\overline{\text{MXSP}}(\cdot)$ are, respectively, a lower bound and an upper bound to $\text{MXSP}(\cdot)$. A simple and feasible implementation, used in this work, is the following:

$$T_{ie} \leq \text{SP}\left(y_e = 1\right) - \textstyle\sum_{e'\in E|a_{ie'}=0} d_{e'} \quad \forall i \in N, e \in E \,|a_{ie} = 1 \tag{37}$$

where SP stands for the classical Shortest Path problem. Therefore, calculating $\underline{\text{MXSP}}(\cdot)$ requires the solution of a SP where arc e is forcibly used.

4.2. On the Value of Υ_e

Constants Υ_e should be greater than the value of γ_e, as it follows from constraints (19). Variables γ_e are associated to the non-negativity of variables y_e. Therefore, their value corresponds to the increase in the shortest path cost resulting from using arc e. We can exploit this definition to devise a lower bound:

$$\Upsilon_e \geq \mathrm{SP}\left(y_e = 1\right) - \mathrm{SP}\left(\right) \quad \forall e \in E \tag{38}$$

Calculating this bound requires the solution of two Shortest Path problems.

5. Decomposition Method for the MXFE

In this section we present the decomposition algorithm for the MXFE. To introduce it, the following notation and terminology is required:

P, Set of paths in graph G, indexed by p.

$E\left(p\right)$, Set of arcs included in path p.

$N\left(p\right)$, Set of nodes interdicting path p. Specifically, $N\left(p\right) = \{i \in N \,|\exists e \in E\left(p\right), a_{i,e} = 1\,\}$.

We call *interdiction pattern* $\overline{\mathbf{x}}$ a vector of binary elements $\overline{x}_i$, one for each node $i \in N$, taking value 1 if a tower is located at i, and 0 otherwise. We say that a path is *covered* if at least one of its arcs is interdicted by a tower. In this sense, a covered path is a path that cannot be traversed. If we could consider all the possible paths existing in graph G, then the problem of maximizing the fire exposure translates into choosing which path to target and, subsequently, maximizing the fire exposure for the targeted path. Necessarily, a path can be targeted only if all the shorter paths have been covered; otherwise, the enemies would follow the shortest uncovered path among them. However, solving this problem would be inefficient even for relatively small graphs, as the number of paths in a graph grows combinatorially. Therefore, we propose a decomposition algorithm that iteratively generates and includes in the model only those paths that are strictly necessary for the solution of the problem. In summary, the steps taken by the algorithm are:

1. Generate the shortest path in the graph obtained by removing from G all the arcs incident to tower locations in the current interdiction pattern (initially empty).
2. Identify the tower configuration that maximizes the fire exposure for the shortest path generated in Step 1, while covering the paths generated in the previous iterations. This step provides a primal solution to the MXFE.
3. Find the interdiction pattern that covers all the paths generated so far, while ensuring at the same time the existence of a path connecting the source node to the sink node. If the problem is unfeasible, stop; the optimal solution to MXFE is the best primal solution found. Otherwise, go to Step 1.

The proposed algorithm is similar in nature to a Benders' decomposition method, although it relies on the solution of three problems rather than just two (i.e., the Master and the Sub problems). The elements comprising the algorithm are presented in the following.

5.1. ISP($\overline{\mathbf{x}}$)

Let $\overline{\mathbf{x}}$ be an interdiction pattern. The ISP($\overline{\mathbf{x}}$) (i.e., Interdicted Shortest Path) identifies the shortest path $\overline{p} \in P$ on a graph where the interdiction of the arcs incident to the nodes in $\overline{\mathbf{x}}$ is enforced:

$$\min \quad \sum_{e \in E} d_e y_e \tag{39}$$

$$s.t. \quad (8), (10) \tag{40}$$

$$y_e = 0 \qquad \forall i \in N, e \in E \,|\overline{x}_i = 1 \wedge a_{ie} = 1 \tag{41}$$

The solution of the ISP can be obtained by the Dijkstra's algorithm (Dijkstra, 1959) run on a reduced graph where all the arcs incident to the locations in $\overline{x}$ have been removed.

5.2. R-MXFE($\overline{P}$,$\overline{p}$)

Let $\overline{P} \subset P$ be a subset of paths and $\overline{p} \in P/\overline{P}$ be a path not included in $\overline{P}$. R-MXFE($\overline{P}, \overline{p}$) (i.e., Reduced MXFE) optimizes the tower configuration so as to maximize the fire exposure of path $\overline{p}$ while covering all the paths in $\overline{P}$:

$$\max \quad \sum_{i \in N} \sum_{k \in K} \sum_{j \in N(\overline{p})} f_{ikj} x_{ik} \tag{42}$$

$$s.t. \quad (2), (3), (6) \tag{43}$$

$$\sum_{i \in N(p)} \sum_{k \in K} x_{ik} \geq 1 \qquad \forall p \in \overline{P} \tag{44}$$

$$x_{ik} = 0 \qquad \forall i \in N\left(\overline{p}\right), k \in K \tag{45}$$

This model maximizes the fire exposure of path $\overline{p}$ (42). Constraints (44) enforce the covering of all the paths in $\overline{P}$, while constraints (45) ensure that path $\overline{p}$ is not covered.

5.3. E-SC($\overline{P}$)

The E-SC($\overline{P}$) (i.e., Extended Set Covering) model is a feasibility model that searches for an interdiction pattern $\overline{x}$ that covers all the paths in $\overline{P}$, while ensuring that there still exists a path connecting the source node s to the sink node t. As this model is concerned exclusively with the locations of the towers, all the parameters regarding the different types of towers have been omitted. Also, the parameter $\overline{R}$ expresses the maximum number of towers that can be located and can be calculated as:

$$\overline{R} = \left\lfloor \frac{R}{\min_{k \in K} \{c_k\}} \right\rfloor. \tag{46}$$

We can now formulate the E-SC($\overline{P}$) as follows:

$$\text{Find} \qquad \overline{\mathbf{x}} \tag{47}$$

$$s.t. \quad \sum_{i \in N} \overline{x}_i \leq \overline{R} \tag{48}$$

$$\sum_{i \in N(p)} x_i \geq 1 \quad \forall p \in \overline{P} \tag{49}$$

$$y_e \leq 1 - \overline{x}_i \qquad \forall i \in N, e \in E \,|a_{ie} = 1 \tag{50}$$

$$\overline{x}_i \in \{0, 1\} \qquad \forall i \in N \tag{51}$$

$$(8), (10) \tag{52}$$

5.4. Decomposition Algorithm

We can now present the decomposition algorithm for the MXFE.

Algorithm 1 Decomposition algorithm for the MXFE.

$\overline{\mathbf{x}} \leftarrow \emptyset$; $\overline{P} \leftarrow \emptyset$;
repeat
  Step 1: $\overline{p} \leftarrow$ arg ISP($\overline{\mathbf{x}}$);
  Step 2: $\hat{\mathbf{x}} \leftarrow$ arg R-MXFE($\overline{P}, \overline{p}$)
  Step 3: if $\hat{\mathbf{x}} > \mathbf{x}^\star$ then $\mathbf{x}^\star \leftarrow \hat{\mathbf{x}}$;
  Step 4: $\overline{P} \leftarrow \overline{P} \cup \{\overline{p}\}$;
  Step 5: $\overline{\mathbf{x}} \leftarrow$ arg E-SC($\overline{P}$);
until E-SC($\overline{P}$) is not feasible;
return $\mathbf{x}^\star$;

The paths subset and the interdiction pattern are initially empty. The main loop of the algorithm solves an ISP to obtain the path $\overline{p}$ followed by the enemy given the current interdiction pattern $\overline{\mathbf{x}}$. Next, a R-MXFE maximizes the fire exposure of the current path, while ensuring that all the paths generated in the previous iterations are covered. In this step, a primal solution, $\hat{\mathbf{x}}$, is obtained and the best solution found, $\mathbf{x}^\star$, is updated if required. Finally, the current path is added to the subset of paths $\overline{P}$ and an interdiction pattern $\overline{\mathbf{x}}$ is obtained by solving an E-SC. The algorithm keeps looping until E-SC is not feasible, meaning that all the paths that can be covered by the player have been generated. The optimal solution to MXFE is given by the best solution found, $\mathbf{x}^\star$.

Theorem 1. *Algorithm 1 correctly solves MXFE.*

We may view Algorithm 1 as follows: Each time the algorithm reaches Step 1, the follower responds as best as possible to the current interdiction pattern $\overline{\mathbf{x}}$ with one new shortest path $\overline{p}$. Then, in Step 5, the leader tries to find a configuration that covers each of the uninterdicted subpaths generated so far. (If such a configuration is found, it may or may not force the follower to traverse a path longer than the path forced by the incumbent solution $\mathbf{x}^\star$.) The algorithm terminates when the leader fails to interdict all the paths in $\overline{P}$ (i.e., E-SC($\overline{P}$) is unfeasible). At this point, the leader cannot force a worse response for the follower than any of the responses seen so far. Therefore, the optimal target path is certainly in $\overline{P}$ and $\mathbf{x}^\star$ is optimal.

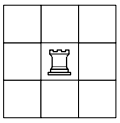
One of the advantages of this algorithm is that the looping condition can be easily changed and the program will always return a feasible solution. For example, in the context considered in this paper, the maximum time available to locate the towers is five seconds. We can stop the computation after five seconds and still get a primal solution. The quality of these heuristic solution is assessed in the following section.

6. Experiments

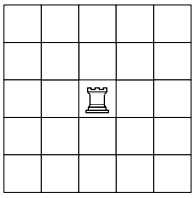
The SL-MXFE has been modeled in GAMS (Brooke et al., 2014) and solved using GLPK (GLPK, 2014), while Algorithm 1 has been written in C++ using GLPK as MIP solver. All the experiments have been run on a computer with an Intel Core i5-2500K CPU having four cores at 3.30GHz and 4GB RAM memory, using only one core per run. To represent the checkerboard structure used in the TD game, we tested the two approaches on a set of grid graphs of different size. The cells are represented by the graph nodes. Only the nodes that correspond to adjacent cells are connected by unitary arcs. Also, we tested for different values of the budget and tower types. Each tower type is described by a tuple <cost, range, fire>, where 'cost' represents the amount of budget required to locate a tower of that type, 'range' identifies the cells affected by the tower, as illustrated in Figure 2, and 'fire' expresses the fire power received by the cells inside the area of effect of the tower. Overall, the parameters considered in the experiments are:

Figure 2: Representation of ranges' areas of effect.

(a) Range 1 area of effect.



(b) Range 2 area of effect.



- Grid size: 3x3, 5x5, 7x7, 9x9, and 11x11.
- Budget R: 1, ..., 10.
- Tower Types <cost, range, fire>: four combinations have been considered.
  1. $\langle 1, 1, 1 \rangle$.
  2. $\langle 1, 1, 1 \rangle$, $\langle 2, 2, 1 \rangle$.
  3. $\langle 1, 1, 1 \rangle$, $\langle 2, 2, 1 \rangle$, $\langle 2, 1, 2 \rangle$.
  4. $\langle 1, 1, 1 \rangle$, $\langle 2, 2, 1 \rangle$, $\langle 2, 1, 2 \rangle$, $\langle 3, 2, 2 \rangle$.
- CPU time: maximum computational time for each run.
  1. 3600 seconds, to assess the overall efficiency of the two proposed methodologies.
  2. 5 seconds, to understand the efficacy of the methodologies when generating heuristic solutions in a real-time context (i.e., when the game is running).

6.1. SL-MXFE: Results

Tables 2 and 3 (in the Appendix) show the results obtained by solving the SL-MXFE with different grid sizes. In the tables, rows are associated to the budget value, R. For each tower type two columns are presented: the solution value and the solution time, respectively. When optimality is proved, a star (*) is placed next to the solution value. When no star appears, the value represents the best solution found. Finally, a dash in the value column means that no feasible solution was found.

The tables show that the SL-MXFE can solve to optimality only the instances having a grid size of 3x3. However, for the 5x5 grid case no optimal solution is found and the model can find a primal solution only for a few problems. Given its poor performance, the model was not tested for graphs larger than 5x5. These results show that formulating the MXFE using the KKT conditions leads to a very inefficient model that cannot be used to solve problems of reasonable size.

6.2. Algorithm 1: Results

Tables 4-8 (in the Appendix) show the results obtained when using Algorithm 1. Each table presents the results for both the 3600 seconds and the 5 seconds experiments. In the latter, the algorithm was run for only 5 seconds and the table reports the value of the best solution found and the associated gap calculated as:

$$GAP = \frac{sol_{3600} - sol_5}{sol_{3600}}$$

where sol_{3600} and sol_5 are the solution values obtained when running the algorithm for 3600 and 5 seconds, respectively. The main results are summarized in the following:

- Grid size 3x3. All problems solved to optimality in less than 0.01 seconds.
- Grid size 5x5. All problems solved to optimality in less than 5 seconds.
- Grid size 7x7. All problems with $R = 1, \ldots, 8$ solved to optimality in approximately 1000 seconds. The algorithm could find the optimal solution within 5 seconds (without proving optimality) in 30 out of 40 experiments.
- Grid size 9x9. All problems with $R = 1, \ldots, 7$ solved to optimality in approximately 600 seconds. The algorithm could find the optimal solution within 5 seconds (without proving optimality) in 28 out of 40 experiments.
- Grid size11x11. All problems with $R = 1, \ldots, 7$ solved to optimality in approximately 1000 seconds. The algorithm could find the optimal solution within 5 seconds (without proving optimality) in 26 out of 40 experiments.

Overall, the algorithm could solve to optimality 168 out of 200 problems in less than 3600 seconds for each problem, and obtained the optimal solution in less than 5 seconds (without proving optimality) in 164 out of 200 problems. Immediately it can be seen that the performances of Algorithm 1 are much better than those of the SL-MXFE. In fact, the algorithm is capable of solving to optimality all the problem instances for grids of size 3x3 and 5x5. In these cases, the solution time for Algorithms 1 is less than 0.01 seconds, only a very small fraction of the time required by the SL-MXFE.

These results show that Algorithm 1 can be successfully applied to a real-time environment as a heuristic method. More generally, we can deduce also that the MXFE lends itself to a heuristic approach. In fact, the optimal solution is generally found at the beginning of the optimization process and most of the solution time is devoted to proving its optimality. Future research could shed some light on the reasons why this happens, and in finding ways to reduce the time required to prove the optimality.

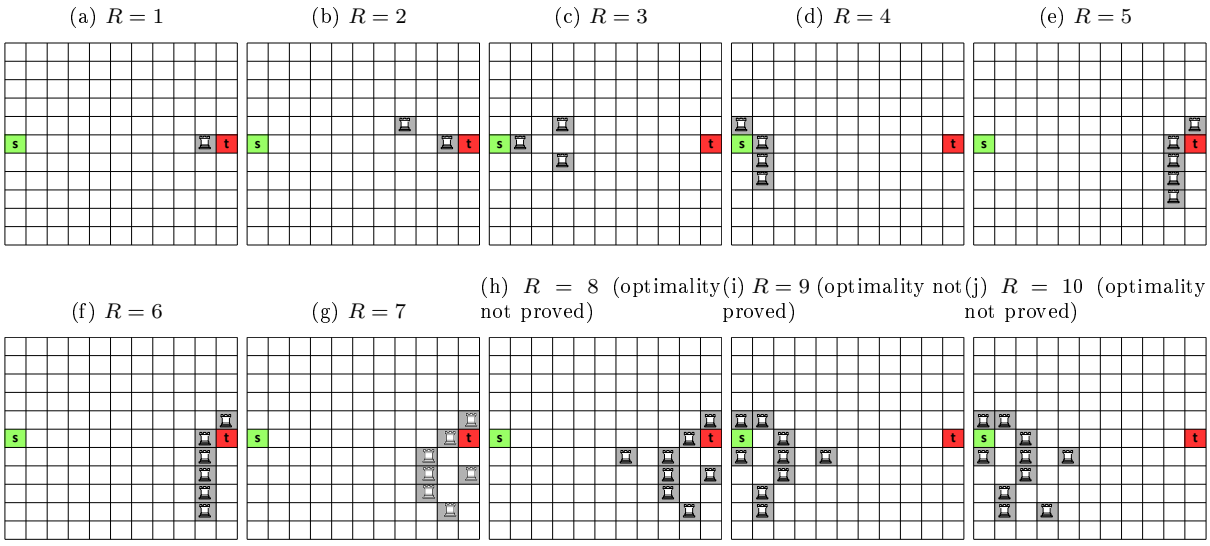
6.3. Grid Size 11x11 Solutions' Layouts

Figure 3 shows the solutions' layouts for the experiments run on a grid of size 11x11 using the tower configuration of type 1 for all the values of $R = 1, \ldots, 10$. Please note that optimality of the solution has not been proved for the cases with $R = 8, 9, 10$. In the figures, the green cell marked with an "s" identifies the enemies source, the red cell marked with a "t" is the enemies sink, and the gray cells with the icon of a tower are the tower locations. By observing the tower configurations a number of interesting insights can be drawn.

First, it is possible to notice that the solutions have a symmetrical quality. In fact, in the instances considered, it is apparently indifferent locating the towers on the right side (next to the source) or on the left side (next to the sink) of the grid. This property of the problem should be studied and understood as future research on this topic could lead to a better understanding of the solution space structure and more efficient models or algorithms.

Second, as the number of towers that can be located increases, the solutions seem to follow a certain pattern, similar to the configurations obtained by cellular automata. A possible line of research could focus on applying statistical or machine learning models that, by analyzing the solutions provided by the MXFE, can infer the rules behind this pattern and provide solutions for big instances that cannot be tackled by Algorithm 1.

Figure 3: Solutions' layout for grid size 11x11 and Tower Type 1.



7. Conclusions

The purpose of this research is to introduce a model for the optimization of tower configurations in a TD game, with the objective of implementing an intelligent controller. The proposed model is bilevel in nature and cannot be solved directly. To deal with the complexity of this problem, we present two optimal methodologies, one based on the KKT conditions and the other on a decomposition reformulation. We show empirically that, while the former can only solve problems of very small size, the latter is very efficient and can tackle large problem instances. Also, we tested the quality of the solutions found with limited computational resources (5 seconds CPU time). Again, the decomposition method proved to be very effective; in fact, it was capable of finding the optimal solution most of the times. Therefore, we can conclude that the proposed decomposition algorithm is suitable for application as part of a commercial videogame, where the computational resources dedicated to decision activities is usually extremely modest.

This work is just a scratch on the surface and several research lines can be pursued, as explained in the following.

- In terms of modeling, the problem could be extended to consider features included in modern TD games, such as different types of enemies and the possibility of destroying existing towers to get resources. An interesting challenge would be the introduction of a time dimension; this would allow to generate dynamic strategies, while, at the same time, increasing significantly the complexity of the model.

- The computational tests revealed that it is relatively simple to find the optimal solution for the MXFE, while is extremely time-consuming to prove its optimality. A possible research path could be the exploration of the causes of this phenomenon. We suspect that it could be related to the high number of multiple optima in the follower level and to the symmetrical nature of the problem. Also, classical heuristic and meta-heuristic methodologies could be applied to the MXFE and their performances compared to that of the decomposition algorithm presented in this paper.

- Knowing the best (or a good) solution to a level allows for the extraction of a number of game measures, such as interestingness (Althöfer, 2003) and uncertainty (Iida et al., 2004). These attributes find

application in the design phase of the development of the game. In fact, they provide a feedback to game designers regarding many aspects of the game. In the academic literature, automated measurements of games have been used to devise Evolutionary Game Design methods (Browne and Maire, 2010), i.e., algorithms capable of designing interesting and challenging games. Similarly, the proposed optimization model for TD games could be the first step toward the definition of a system that automatically generates levels and game elements, according to a specified level of challenge. Additionally, the solutions identified by the model can be used as a basis of comparison for the players' actions while the game is running. This, in turn, allows for further analyses, such as assessing their level of skill and cooperation.

We hope this article will motivate many researchers in OR to tackle more problems in the context of TD games and videogames in general, and to extend the application of OR techniques in these rich and exciting fields.

Acknowledgments

The research of Liberatore was supported by the Spanish Government grant TIN2012-32482 and the U.S.-Italy Fulbright Commission (Fulbright-Finmeccanica grant). The research of Camacho has been partially supported by Savier, an Airbus Defense & Space project (FUAM-076914), and by the CIBERDINE project (S2013/ICE-3095). Also, Liberatore would like to thanks the AIDA research group for the kind hospitality during his visit. All the supports are gratefully acknowledged.

References

M. Nayak, L. Baker, Factbox: A look at the $78 billion video games industry, Reuters .

B. Colson, P. Marcotte, G. Savard, An overview of bilevel optimization, Annals of Operations Research 153(1) (2007) 235–256.

W. Karush, Minima of Functions of Several Variables with Inequalities as Side Constraints, Master's thesis, Deptartment of Mathematics, University of Chicago, 1939.

H. Kuhn, A. Tucker, Nonlinear programming, in: Proceedings of 2nd Berkeley Symposium, University of California Press, 481–492, 1951.

C. Browne, F. Maire, Evolutionary Game Design, IEEE Transactions on Computational Intelligence and AI in Games 2(1) (2010) 1–16.

P. Avery, J. Togelius, E. Alistar, R. van Leeuwen, Computational intelligence and tower defence games, in: Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC), IEEE, 1084–1091, 2011.

P. Huo, S. Shiu, H. Wang, B. Niu, Application and Comparison of Particle Swarm Optimization and Genetic Algorithm in Strategy Defense Game, in: Proceedings of the Fifth International Conference on Natural Computation (ICNC '09), vol. 5, IEEE, 387–392, 2009.

T. G. Tan, Y. N. Yong, K. O. Chin, J. Teo, R. Alfred, Soft Computing Applications and Intelligent Systems, chap. "Automated Evaluation for AI Controllers in Tower Defense Game Using Genetic Algorithm", Springer, 135–146, 2013.

P. Rummell, Adaptive AI to play tower defense game, in: Proceedings of the 16th International Conference on Computer Games (CGAMES), IEEE, 38–40, 2011.

R. Wollmer, Removing arcs from a network, Operations Research 12(6) (1964) 934–940.

R. Church, M. Scaparra, R. Middleton, Identifying critical infrastructure: the median and covering facility interdiction problem, Annals of the Association of American Geographers 94(3) (2004) 491–502.

C. Losada, M. Scaparra, R. Church, M. Daskin, The multiple resource probabilistic interdiction median problem, Tech. Rep., KBS Working Paper n. 187, University of Kent, 2009.

D. Granata, G. Steeger, S. Rebennack, Network interdiction via a Critical Disruption Path: Branch-and-Price algorithms, Computers and Operations Research 40(11) (2013) 2689–2702.

T. Matisziw, A. Murray, Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure, Computers and Operations Research 36(1) (2009) 16–26.

E. Israeli, R. Wood, Shortest-Path Network Interdiction, Networks 40(2) (2002) 97–111.

D. Aksen, S. Akca, N. Aras, A bilevel partial interdiction problem with capacitated facilities and demand outsourcing, Computers and Operations Research 41(1) (2014) 346–358.

H. von Stackelberg, Market Structure and Equilibrium, Springer, 2011.

M. Scaparra, R. Church, A bilevel mixed-integer program for critical infrastructure protection planning, Computers and Operations Research 35(6) (2008a) 1905–1923.

M. Scaparra, R. Church, An exact solution approach for the interdiction median problem with fortification, European Journal of Operational Research 189(1) (2008b) 76–92.

D. Kamphausen, M. Giersch, TowerDefense, https://github.com/godrin/TowerDefense, 2012.

E. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1(1) (1959) 269–271.

A. Brooke, D. Kendrick, A. Meeraus, R. Raman, GAMS - A User's Guide, GAMS Development Corporation, 2014.

GLPK, GNU Linear Programming Kit, http://www.gnu.org/software/glpk/, 2014.

I. Althöfer, Computer-aided game inventing, Tech. Rep., Friedrich Schiller Universität Jena, 2003.

H. Iida, K. Takahara, J. Nagashima, Y. Kajihara, T. Hashimoto, An application of game-refinement theory to Mah Jong, in: M. Rauterberg (Ed.), Entertainment Computing - ICEC 2004, vol. 3166, Springer, 445–450, 2004.

Appendix

Table 2: SL-MXFE, grid size 3x3, CPU time limit 3600 seconds.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	5*	1	5*	1	5*	1.822	5*	1.785
2	7*	1	7*	1	10*	1	10*	6.871
3	8*	1	10*	1	12*	15.127	12*	1
4	10*	1.22	12*	1	15*	1	15*	244.745
5	12*	1	15*	16.352	17*	78.375	20*	351.411
6	14*	0.3	17*	1	20*	1	22*	1
7	14*	0.346	20*	1.838	22*	7.16	25*	243.305
8	14*	1	20*	2.242	25*	7.455	30*	1
9	14*	1	20*	1	25*	1	32*	1
10	14*	1	20*	1	25*	1	35*	20.553

Table 3: SL-MXFE, grid size 5x5, CPU time limit 3600 seconds.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	-	3600	-	3600	-	3600	-	3600
2	-	3600	-	3600	-	3600	-	3600
3	-	3600	-	3600	-	3600	-	3600
4	-	360	-	360	-	360	-	360
5	18	3600	-	3600	-	3600	-	3600
6	18	3600	-	3600	-	3600	-	3600
7	13	3600	20	3600	-	3600	-	3600
8	13	3600	29	3600	-	3600	-	3600
9	26	3600	-	3600	-	3600	-	3600
10	-	3600	-	3600	-	3600	-	3600

Table 4: Algorithm 1, grid size 3x3.

(a) Experiments with CPU time 3600 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	5*	0	5*	0	5*	0	5*	0
2	7*	0	7*	0	10*	0	10*	0
3	8*	0	10*	0	12*	0	12*	0
4	10*	0	12*	0	15*	0	15*	0
5	12*	0	15*	0	17*	0	20*	0
6	14*	0	17*	0	20*	0	22*	0
7	14*	0	20*	0	22*	0	25*	0
8	14*	0	20*	0	25*	0	30*	0
9	14*	0	20*	0	25*	0	32*	0
10	14*	0	20*	0	25*	0	35*	0

(b) Experiments with CPU time 5 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	GAP	Value	GAP	Value	GAP	Value	GAP
1	5*	0%	5*	0%	5*	0%	5*	0%
2	7*	0%	7*	0%	10*	0%	10*	0%
3	8*	0%	10*	0%	12*	0%	12*	0%
4	10*	0%	12*	0%	15*	0%	15*	0%
5	12*	0%	15*	0%	17*	0%	20*	0%
6	14*	0%	17*	0%	20*	0%	22*	0%
7	14*	0%	20*	0%	22*	0%	25*	0%
8	14*	0%	20*	0%	25*	0%	30*	0%
9	14*	0%	20*	0%	25*	0%	32*	0%
10	14*	0%	20*	0%	25*	0%	35*	0%

Table 5: Algorithm 1, grid size 5x5.

(a) Experiments with CPU time 3600 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	4*	0	4*	0	4*	0	4*	0
2	8*	0	8*	0.01	8*	0	8*	0
3	12*	0.01	12*	0.01	14*	0.01	14*	0.01
4	17*	0.06	17*	0.07	19*	0.07	20*	0.07
5	21*	0.36	21*	0.36	24*	0.36	26*	0.37
6	26*	1.51	26*	1.52	29*	1.52	32*	1.53
7	32*	2.51	32*	2.51	36*	2.53	38*	2.51
8	36*	3.59	37*	3.59	41*	3.59	44*	3.58
9	37*	3.82	42*	3.85	46*	3.84	51*	3.83
10	38*	3.83	46*	3.86	51*	3.85	57*	3.86

(b) Experiments with CPU time 5 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	GAP	Value	GAP	Value	GAP	Value	GAP
1	4*	0%	4*	0%	4*	0%	4*	0%
2	8*	0%	8*	0%	8*	0%	8*	0%
3	12*	0%	12*	0%	14*	0%	14*	0%
4	17*	0%	17*	0%	19*	0%	20*	0%
5	21*	0%	21*	0%	24*	0%	26*	0%
6	26*	0%	26*	0%	29*	0%	32*	0%
7	32*	0%	32*	0%	36*	0%	38*	0%
8	36*	0%	37*	0%	41*	0%	44*	0%
9	37*	0%	42*	0%	46*	0%	51*	0%
10	38*	0%	46*	0%	51*	0%	57*	0%

Table 6: Algorithm 1, grid size 7x7.

(a) Experiments with CPU time 3600 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	4*	0	4	0*	4	0*	4	0*
2	9*	0.02	9	0.02*	9	0.03*	9	0.03*
3	12*	0.04	12	0.04*	14	0.04*	14	0.04*
4	18*	0.3	18	0.31*	19	0.31*	19	0.33*
5	23*	2.17	23	2.26*	24	2.33*	24	2.35*
6	26*	13.45	27	13.98*	30	14.32*	33	14.33*
7	30*	100.61	32	102.58*	36	104.8*	40	103.97*
8	38*	1035.79	38	1046.94*	41	1060.83*	48	1055.46*
9	45	3600	46	3600	46	3600	53	3600
10	53	3600	53	3600	53	3600	60	3600

(b) Experiments with CPU time 5 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	GAP	Value	GAP	Value	GAP	Value	GAP
1	4*	0%	4*	0%	4*	0%	4*	0%
2	9*	0%	9*	0%	9*	0%	9*	0%
3	12*	0%	12*	0%	14*	0%	14*	0%
4	18*	0%	18*	0%	19*	0%	19*	0%
5	23*	0%	23*	0%	24*	0%	24*	0%
6	26*	0%	27*	0%	30*	0%	33*	0%
7	30*	0%	32*	0%	36*	0%	40*	0%
8	33	13%	37	3%	41*	0%	48*	0%
9	37	18%	43	7%	45	2%	53	0%
10	45	15%	48	9%	49	8%	60	0%

Table 7: Algorithm 1, grid size 9x9.

(a) Experiments with CPU time 3600 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	4*	0	4*	0.01	4*	0.01	4*	0
2	9*	0.05	9*	0.06	9*	0.06	9*	0.06
3	12*	0.11	12*	0.12	13*	0.11	13*	0.13
4	18*	0.79	18*	0.89	19*	0.83	19*	0.86
5	23*	4.88	23*	5.4	25*	5.1	25*	5.2
6	29*	45	30*	49.71	31*	46.96	32*	47.44
7	32*	579.88	35*	615.44	37*	596.8	42*	599.78
8	37	3600	41	3600	43	3600	49	3600
9	41	3600	46	3600	48	3600	56	3600
10	47	3600	52	3600	54	3600	64	3600

(b) Experiments with CPU time 5 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	GAP	Value	GAP	Value	GAP	Value	GAP
1	4*	0%	4*	0%	4*	0%	4*	0%
2	9*	0%	9*	0%	9*	0%	9*	0%
3	12*	0%	12*	0%	13*	0%	13*	0%
4	18*	0%	18*	0%	19*	0%	19*	0%
5	23*	0%	23*	0%	25*	0%	25*	0%
6	29*	0%	30*	0%	31*	0%	32*	0%
7	32*	0%	35*	0%	37*	0%	42*	0%
8	36	3%	39	5%	43	0%	49	0%
9	39	5%	45	2%	48	0%	56	0%
10	42	11%	50	4%	53	2%	64	0%

Table 8: Algorithm 1, grid size 11x11.

(a) Experiments with CPU time 3600 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)
1	4*	0.01	4*	0.01	4*	0.01	4*	0.01
2	9*	0.13	9*	0.13	9*	0.13	9*	0.14
3	12*	0.14	12*	0.14	13*	0.14	13*	0.17
4	18*	0.95	18*	0.99	19*	0.99	19*	1.02
5	24*	7.44	24*	7.6	25*	7.67	25*	7.79
6	30*	68.21	30*	69.56	31*	70.12	33*	70.15
7	35*	1069.25	35*	1058.75	37*	1066.84	42*	1070.29
8	38	3600	41	3600	41	3600	49	3600
9	43	3600	49	3600	49	3600	56	3600
10	47	3600	56	3600	56	3600	64	3600

(b) Experiments with CPU time 5 s.

	Tower Type 1		Tower Type 2		Tower Type 3		Tower Type 4	
R	Value	GAP	Value	GAP	Value	GAP	Value	GAP
1	4*	0%	4*	0%	4*	0%	4*	0%
2	9*	0%	9*	0%	9*	0%	9*	0%
3	12*	0%	12*	0%	13*	0%	13*	0%
4	18*	0%	18*	0%	19*	0%	19*	0%
5	24*	0%	24*	0%	25*	0%	25*	0%
6	29	3%	30*	0%	31*	0%	33*	0%
7	32	9%	35*	0%	37*	0%	42*	0%
8	34	11%	38	7%	41	0%	49	0%
9	38	12%	45	8%	48	2%	56	0%
10	44	6%	50	11%	55	2%	62	3%