

# Heuristic Algorithms for Finding Irregularity Strengths of Graphs

David K. Garnick<sup>1</sup> and Jeffrey H. Dinitz<sup>2</sup>

**Abstract:** Given a graph  $G$  with weighting  $w : E(G) \rightarrow Z^+$ , the *strength* of  $G(w)$  is the maximum weight on any edge. The *sum* of a vertex in  $G(w)$  is the sum of the weights of all its incident edges.  $G(w)$  is *irregular* if the vertex sums are distinct. The *irregularity strength* of a graph is the minimum strength of the graph under all irregular weightings. We present fast heuristic algorithms, based on hill-climbing and simulated annealing, for finding irregular weightings of a given strength. The heuristics are compared empirically, and the algorithms are then used to formulate a conjecture.

## 1 Introduction

Because of the immense size of the search space, most combinatorial problems do not lend themselves to exhaustive search algorithms. In many cases, however, the use of local search or hill-climbing algorithms provide a useful alternative to exhaustive search. In a hill-climbing algorithm (as in all optimization problems) a cost function is defined on the points in a search space. Heuristics are then employed which move the instance around the search space without (in general) increasing the cost. The hope is that, with many applications of the heuristics, a point in the search space with an acceptably low cost can be found in a very short time.

Hill-climbing algorithms have proven successful in a number of combinatorial optimization problems. Local search techniques have been applied to the traveling salesman problem [Li], minimum cost survivable networks [SWK], to the construction of offshore natural gas pipeline systems [RFRSK] and to the problem of uniform graph partitioning [KL1]. In each of these cases, a solution is found in *reasonable* time which gives a local optimum for the cost function. A desirable local optimum will be sufficiently close in value to the global optimum for this cost function. [PS] discusses local search techniques in the context of optimization problems.

In combinatorial design theory, one wishes to construct subsets (called *blocks*) of a set. The blocks must satisfy certain prescribed constraints such as constant size and pairwise balance of elements. A block design is constructed by adding new blocks to an existing partial design. If no new blocks can be added, then a hill-climbing algorithm will typically delete a block in order to add a new block. The cost of a partial design is defined to be the number of blocks missing from

---

<sup>1</sup>Department of Computer Science, Bowdoin College, Brunswick, ME 04011, USA. This author's research was supported by the National Science Foundation under Grant CCR-8909357.

<sup>2</sup>Department of Mathematics, University of Vermont, Burlington, VT 05405, USA.

the partial design. Thus, the algorithm succeeds only when a design is found with a cost of zero; in this sense, a combinatorial design problem is much more restrictive than many optimization problems. Even with such a restrictive definition of success, hill-climbing algorithms have proven to be extremely useful. In design theory, hill-climbing algorithms have been successfully used to construct Steiner triple systems [S], Strong starters [DS1], one-factorizations of  $K_n$  [DS3] and Room squares [DS3]. In each of these instances an exhaustive search is impractical when the size of the design gets even moderately large due to the enormous size of the search space. Yet local search not only succeeds, but succeeds quickly. It is our hope that showing the usefulness of local search techniques in the context of the irregularity strength of a graph will help to popularize these techniques for other problems as well.

Based on discussion in [PS], we now review the general idea behind hill-climbing. Given an instance  $(F, c)$  of an optimization problem, where  $F$  is a feasible set and  $c$  is a cost function, choose a neighborhood,  $N : F \rightarrow 2^F$ , which is searched at a point  $t$  in  $F$  for improvements by heuristics selected from a set  $H$ . The general form of  $h \in H$  is

$$h(t) = \begin{cases} \text{an } s \in N(t) \text{ with } c(s) \leq c(t) & \text{if such an } s \text{ exists} \\ t & \text{otherwise} \end{cases}$$

Typically, when starting a hill-climbing search for a design, the initial partial design is the empty design. In general, when partial design  $s$  can be constructed from a partial design  $t$  with  $s$  having no fewer blocks than  $t$ , then the partial design  $s$  is adopted and the search proceeds from  $s$ . The general hill-climbing algorithm is given below.

```

algorithm hill-climbing
   $t \leftarrow$  the empty design (or other starting partial design)
  while  $cost(t) \neq 0$  do
    choose  $h \in H$ 
     $t \leftarrow h(t)$ 
[PS]

```

The art in designing the algorithm is in finding appropriate heuristics which in general do not return the original point  $t$ . When more than one heuristic is defined, the heuristic chosen at each iteration can be selected randomly (with some probability), or on the basis of properties of the point  $t$ . Note also that the hill-climbing algorithm may fail to produce a solution to the problem. Thus, it is essential that the algorithm have a reasonable probability of finding a solution in an acceptable amount of time.

## 1.1 The Problem

We now describe the graph problem on which we will use our local search heuristics. Let  $G = (V, E)$  be a simple graph with no  $K_2$  component and at most one isolated vertex. A network  $G(w)$  consists

of the graph  $G$  together with an assignment  $w : E(G) \rightarrow Z^+$ . The *strength*  $s$  of  $G(w)$  is defined by  $s(G(w)) = \max\{w(e) : e \in E(G)\}$ . For each vertex  $v \in V(G)$ , define the *sum*  $\sigma(v)$  of  $v$  in  $G(w)$  by  $\sum_{e \text{ incident to } v} w(e)$  and call  $G(w)$  *irregular* if for all distinct  $u, v \in V(G)$ ,  $\sigma(u) \neq \sigma(v)$ . The *irregularity strength*  $I(G)$  is defined to be  $\min\{s(G(w)) : G(w) \text{ is irregular}\}$ . Thus the irregularity strength of a graph  $G$  is the smallest strength of all irregular weightings of  $G$ . The problem can also be described as that of choosing positive weights for the non-zero entries in an adjacency matrix such that the row sums are distinct. [FJLS]

Figure 1 shows an irregular weighting of strength 5 on the Petersen graph, which [CJLORS] showed to be the irregularity strength of that graph. (The vertices are labeled with their respective sums.)

Figure 1: Irregular weighting on the Petersen graph

[CJLORS] proposed studying  $I(G)$ . They showed that  $(3p - 2q)/3 \leq I(G) \leq 2p - 3$  for a graph  $G$  with  $p$  vertices and  $q$  edges. In [JL] a stronger lower bound was obtained:

$$I(G) \geq \lambda(G) = \max \left\{ \left( \binom{j}{\sum_{k=i}^j d_k} + i - 1 \right) / j : i \leq j \right\}$$

where  $d_k$  is the number of vertices of degree  $k$  in  $V(G)$ .

The problem of studying irregularity strengths of graphs has proven to be fairly difficult. There are not a great many graphs for which the irregularity strength is known. [CJLORS] showed that  $I(K_n) = 3$  and  $I(K_{2n,2n}) = 3$ ;  $I(P_n)$  was also determined. [G] showed that  $I(K_{2n+1,2n+1}) = 4$ . Work has also been done on binary trees, dense graphs, and the disjoint unions of paths, cycles and complete graphs ([CSS], [FJKL], [KL2]). Recently, [EHLW1] determined the irregularity strengths of wheels,  $k$ -cubes and  $2 \times n$  grids. In each of these cases they found that  $I(G) = \lambda(G)$  or  $\lambda(G) + 1$

and they conjecture that if  $T$  is a tree, then  $I(T) = \lambda(T)$  or  $\lambda(T) + 1$ . [L] surveys results on irregularity strengths of graphs.

## 1.2 Overview of Paper

In addition to hill-climbing, we will discuss the effectiveness of a related simulated annealing heuristic. In brief, simulated annealing is a form of local search in which the heuristics are allowed, with some probability, to increase the value of the cost function. Typically, the probability decreases as the cost decreases. This method is discussed in [KGV] and has proven to be useful in some combinatorial optimization problems (see [W] for instance).

Section 2 describes our set of heuristics, and section 3 presents the algorithms for irregularly weighting graphs. Section 4 gives some empirical results on the effectiveness of the algorithms and section 5, the conclusion, shows their use in formulating a conjecture on a specific class of graphs.

## 2 The Heuristics

This section presents the set of heuristics on which the algorithms are based. The heuristics provide methods for assigning weights to unweighted edges, and changing the weights on previously weighted edges. We first provide some definitions. For a graph  $G$ ,

1. For all  $v \in V(G)$ ,  $v$  is *completed* if all  $(u, v) \in E(G)$  are weighted. By extension,  $G$  is completed, or *completely weighted*, if all  $e \in E(G)$  are weighted.
2. For all completed distinct  $u, v \in V(G)$ ,  $u$  and  $v$  *conflict* if  $\sigma(u) = \sigma(v)$ .
3. The *cost* of a partial weighting without conflicts is the number of unweighted edges.

Each heuristic is designed to preserve the following properties:

1. For all weighted  $e \in E(G)$ ,  $1 \leq w(e) \leq s$  where  $s$  is the strength of the attempted weighting;
2. For all completed  $u, v \in V(G)$ , if  $\sigma(u) = \sigma(v)$  then  $u = v$ .

We use four heuristics which preserve these properties: Climb, Scan, Anneal, and Rank.

### 2.1 Climb Heuristic

The Climb heuristic randomly selects an unweighted edge  $e = (t, u)$ . The set  $\{1, 2 \dots s\}$  is then partitioned into three mutually disjoint sets  $W_0, W_1$ , and  $W_2$ , where  $W_i$  contains the weights that will cause  $i$  of the vertices in  $\{t, u\}$  to be in conflict with some vertex in  $V(G)$ . If  $w \in \{1, 2 \dots s\}$  causes  $t$  and  $u$  to conflict only with each other,  $w$  is put in  $W_1$ .

**procedure** Climb

$e \leftarrow$  randomly selected unweighted edge  $(t, u) \in E(G)$   
partition  $\{1, 2 \dots s\}$  into  $W_0, W_1$ , and  $W_2$  based on  $e$   
if  $W_0 \neq \phi$  then  $weight(e) \leftarrow$  a random  $w \in W_0$   
else if  $W_1 \neq \phi$  then  
     $weight(e) \leftarrow$  a random  $w \in W_1$   
     $v \leftarrow$  the vertex that conflicts with a vertex in  $\{t, u\}$   
    unweight a random edge incident on  $v$

As is typical with climbing heuristics, a sideways step is permissible if the local search does not yield a point with a smaller value for the cost function. However, downhill steps are not permitted; choosing a weight from  $W_2$  is likely to require unweighting two edges. Thus, Climb takes no action when  $W_0 = W_1 = \phi$ .

By maintaining an array which maps sums onto completed vertices, we are able to determine in constant time the number of conflicts that would occur by assigning a particular weight to a given edge. The time complexity of the heuristic is governed by the number of weights partitioned into the  $W$  sets. Therefore, the time complexity of Climb is  $O(s)$ .

## 2.2 Scan Heuristic

Climb is an example of a strict hill-climbing heuristic; it does not permit the cost function to grow, and is restricted in the ways it can move the partial solution about the search space. Thus, the Climb heuristic can become trapped in a local optimum. Since the only acceptable solutions are those with a zero cost, the algorithm must have a heuristic to escape local optima.

To extend the metaphor of hill-climbing, we use another heuristic to *scan* the horizon of the solution space for a point that has an altitude equal to that of the current partial solution. If such a point can be found, the algorithm hops directly to that point in the hope that it is on the slope of a hill with a higher peak. (Since we are using the metaphor of hill-climbing, an *uphill* step reduces the cost function and approaches a solution. A *downhill* step increases the cost function and is a step away from a complete solution.) The following is the Scan heuristic.

**procedure** Scan

$e \leftarrow$  randomly selected weighted edge in  $E(G)$   
from  $\{1, 2 \dots s\}$ , form  $W_0$  based on  $e$   
if  $W_0 - weight(e) \neq \phi$  then  
     $weight(e) \leftarrow$  a random element from  $W_0 - weight(e)$

Scan attempts to reweight a weighted edge with a new weight that will not cause either of the edge's endpoints to conflict with other vertices in the graph. Again, the time complexity of this heuristic is governed by the partitioning of the set of weights. Thus, Scan is  $O(s)$ .

## 2.3 Anneal Heuristic

Simulated annealing is similar to hill-climbing; where hill-climbing uses a scanning heuristic to escape local optima, simulated annealing allows an occasional downhill step. Simulated annealing uses a *cooling schedule* to regulate the probability with which the algorithm will take a downhill step (or, using the metaphor of annealing, *jump to a higher energy level*). According to the fixed schedule, the state is eventually frozen and the current local optimum is accepted. Since we are searching for global optima, we do not consider the state frozen until the algorithm finds a state with a zero cost. In the Anneal heuristic, the probability  $p$  can be calculated as a function of the cost of the current partial weighting, or can be fixed for the duration of the algorithm. The pseudo-code “with probability  $p$ ” can be interpreted as a function which returns **true** with a probability of  $p$ , and otherwise returns **false** .

```

procedure Anneal
   $e \leftarrow$  randomly selected unweighted edge  $(t, u) \in E(G)$ 
  partition  $\{1, 2 \dots s\}$  into  $W_0, W_1$ , and  $W_2$  based on  $e$ 
  if  $W_0 \neq \phi$  then  $weight(e) \leftarrow$  a random  $w \in W_0$ 
  else if  $(W_2 \neq \phi)$  and (with probability  $p$ ) then
     $weight(e) \leftarrow$  a random  $w \in W_2$ 
     $v_1 \leftarrow$  the vertex that conflicts with  $t$ 
    unweight a random edge incident on  $v_1$ 
    if  $\exists v_2 \in V(G)$  that conflicts with  $u$  then
      unweight a random edge incident on  $v_2$ 
  else if  $W_1 \neq \phi$  then
     $weight(e) \leftarrow$  a random  $w \in W_1$ 
     $v \leftarrow$  the vertex that conflicts with a vertex in  $\{t, u\}$ 
    unweight a random edge incident on  $v$ 

```

This heuristic is an adaptation of the Metropolis algorithm [MRRTT]. The Metropolis algorithm permits a downhill step only when no uphill or sideways steps are available. In our algorithms, such a heuristic sometimes leads to an infinite loop of sideways steps between two neighboring points in the solution space. Thus, with some probability, we take a downhill step even when a sideways step is possible. An alternative would be to use the Metropolis algorithm in conjunction with some other form of sidestepping (such as Scan).

## 2.4 Rank Heuristic

We define the *rank* of an edge  $e = (u, v) \in E(G)$  to be  $r(e) = degree(u) + degree(v)$ . Rank is a *greedy* heuristic in that it reserves large weights for edges with high rank, since the endpoints of such edges will tend to have large sums in a completed irregular weighting, and small weights for edges with low rank. *RankCount* is an array where

$$RankCount[i] = |\{e : e \in E(G), r(e) < i\}|$$

```

procedure Rank
  for  $i \leftarrow 1$  to  $|E(G)|$  do
     $weight(e_i = (t, u)) \leftarrow \lfloor RankCount[r(e_i)] * s / (|E(G)| + 1) \rfloor + 1$ 
    if  $\exists v_1 \in V(G)$  that conflicts with  $t$  then unweight a random edge incident on  $v_1$ 
    if  $\exists v_2 \in V(G)$  that conflicts with  $u$  then unweight a random edge incident on  $v_2$ 

```

Figure 2 shows a graph after a single application of Rank. The edges of lower rank were assigned lower weights. The bottom edge was initially assigned a weight of 2; however, weighting the upper horizontal edge with 3 created a conflict between 2 vertices (both of which had vertex sums of 4). The conflict was resolved by unweighting the bottom edge.

Figure 2: A graph after an application of Rank

Rarely can a single application of Rank successfully weight all of the edges in a graph, and experimental evidence indicates that it is not worthwhile to repeatedly use Rank. However, Rank often generates a partial weighting that is close to some complete irregular weighting. Thus, Rank is useful for generating a partial weighting as input to the other heuristics.

The rank of each edge is determined in constant time, and the RankCount array is computed in time proportional to  $O(|E(G)|)$ . Using the array described in Climb, conflicts at each step are detected and resolved in constant time. Therefore Rank is  $O(|E(G)|)$ .

### 3 The Algorithms

This section presents our algorithms for finding irregular weightings of strength  $s$  on a graph  $G$ . If such an algorithm has made a large number of applications of heuristics without finding a complete irregular weighting, one may reasonably conclude that the current partial weighting is a long way down a blind alley. (Of course, the very next application of a heuristic *might* complete the weighting.) We therefore find it useful to define a threshold function,  $T(G)$ , in order to force an algorithm to unweight some edges in  $G$  if  $T(G)$  applications of heuristics have been made without yielding a completed weighting. The algorithm then resumes applying heuristics, after resetting the count toward the threshold to zero. Since an algorithm can quickly get to within relatively few edges of a complete weighting, we choose to unweight all edges whenever the threshold is reached.

If the threshold function is well chosen, we have found that  $n$  tries of  $T(G)$  steps are more likely to yield a solution than one try of  $nT(G)$  steps.

Each of the following algorithms repeatedly selects and applies heuristics to partial weightings. However, in all cases the Rank heuristic is applied only when all edges are unweighted; either at the start of the algorithm or when the threshold has been attained.

The input to all of the algorithms is a strength  $s$ , and an undirected graph  $G$  with all edges unweighted. The algorithms have additional algorithm-specific parameters. The *Cost* always refers to the current number of unweighted edges. If the algorithm terminates on a given instance, its output is  $w : E(G) \rightarrow \{1, 2, \dots, s\}$  such that  $G(w)$  is irregular.

### 3.1 Irregularity Strength 1 (IS-1)

The general strategy behind this algorithm is to repeatedly apply the Climb heuristic until some number of consecutive applications have not yielded an improvement (that is, a decrease in cost). At that point the Scan heuristic is applied for a number of times before resuming Climbing. All edges are unweighted when the threshold is attained, and Rank may be performed at that time. There are three parameters to this algorithm:

1. *Threshold* The number of heuristic steps required to trigger unweighting all edges.
2. *ClimbFactor* A factor used in determining the number of unsuccessful Climbs required to trigger scanning.
3. *ScanQuotient* A quotient used in determining the number of times Scan is applied.

algorithm IS-1

```
optionally perform Rank
Steps ← 0
repeat
  Stuck ← 0
  OldCost ← Cost
  repeat
    Climb; Steps ← Steps + 1
    if Cost = OldCost then Stuck ← Stuck + 1
    else
      OldCost ← OldCost - 1
      Stuck ← 0
  until (Cost = 0) or (Stuck ≥ Cost * ClimbFactor) or (Steps ≥ Threshold)
  if (Cost > 0) then
    for i ← 1 to (|E(G)| - Cost) / ScanQuotient do
      Scan; Steps ← Steps + 1
  if (Cost > 0) and (Steps ≥ Threshold) then
    unweight all edges
    optionally perform Rank
    Steps ← 0
until Cost = 0
```

### 3.2 Irregularity Strength 2 (IS-2)

The generic description of hill-climbing in section 1 includes a step where the algorithm chooses the next heuristic to apply. Algorithm IS-1 makes the decision between Climb and Scan on the basis of the state of the computation. In contrast, IS-2 randomly selects (probabilistically) either Climb or Scan. At each step, algorithm IS-2 applies Climb with a probability  $p$ , and Scan with a probability of  $1 - p$ . As in IS-1, all edges are unweighted when the threshold is attained, and Rank may be performed when all edges are unweighted.

algorithm IS-2

```
optionally perform Rank
Steps ← 0
repeat
  if with probability  $p$  then Climb
  else Scan
  Steps ← Steps + 1
  if (Cost > 0) and (Steps ≥ Threshold) then
    unweight all edges
    optionally perform Rank
    Steps ← 0
until Cost = 0
```

### 3.3 Irregularity Strength 3 (IS-3)

IS-3 is similar in design to IS-2; however, Anneal is the only heuristic applied by IS-3. The parameter  $p$  is used by Anneal to determine when a downhill step is permissible.

```
algorithm IS-3
  optionally perform Rank
  Steps  $\leftarrow$  0
  repeat
    Anneal; Steps  $\leftarrow$  Steps + 1
    if (Cost > 0) and (Steps  $\geq$  Threshold) then
      unweight all edges
      optionally perform Rank
      Steps  $\leftarrow$  0
  until Cost = 0
```

## 4 Empirical Results

Two of the major factors that determine the irregularity strength of a graph are its regularity and the density of its edges. These factors guided our choice of classes of graphs for applying the algorithms; we chose  $K_n$ ,  $X_n$  (the class of  $n \times n$  grids), and random graphs in which each possible edge appears with a probability of .5 ( $|E(G)| \approx |V| * (|V| - 1)/4$ ).

On each graph we gave each algorithm (under various parameters) 100 *tries*, where a try consists of running the algorithm until either a complete irregular weighting was found, or the threshold was reached for the first time. For each algorithm and set of parameters we recorded the frequency with which tries were successful, and the average number of  $O(s)$  operations used during successful tries. (These are labeled *succ rate* and *ops/succ* in tables 1-3.) The figures in the tables do not include the single  $O(|E(G)|)$  step included when Rank was performed. In all tries we used  $\lceil \lambda(G) \rceil$  as the strength of the attempted irregular weighting.

Tables 1-3 show the results of applying these algorithms to  $X_{10}$  using strength of  $I(X_{10}) = 26$ . (Since there are 4 vertices of degree 2,  $4(n - 2)$  vertices of degree 3, and  $(n - 2)^2$  vertices of degree 4 in  $X_n$ , then  $\lambda(X_n) = (n^2 + 1)/4$ .) A threshold of 50,000 was used for all of these experiments. Similar results were obtained for several graphs in  $X_n$ ,  $K_n$ , and random graphs as described above. These results are reported in [GD].

As a basis for comparison, we applied a simple backtracking algorithm to small grids in  $X_n$  using  $\lambda(X_n)$  as the strength. The number of  $O(s)$  operations required to find the first solution is 4 operations for  $X_2$ , 372 operations for  $X_3$ , and 256,670 operations for  $X_4$ ; the backtrack algorithm applied to  $X_5$  was halted, prior to finding a solution, after 213,000,000 operations. Since  $|E(X_n)| = 2(n^2 - n)$ , the complexity of backtracking applied to  $X_n$  is  $O(s^{(n^2)})$ . We note that using

ClimbFactor		1			4			16		
ScanQuotient		1	4	16	1	4	16	1	4	16
no	succ rate	.10	.33	.62	.24	.53	.49	.44	.38	.42
Rank	ops/succ	36001	27510	22070	27442	22161	19107	21379	19600	19433
with	succ rate	.11	.38	.61	.32	.56	.71	.50	.64	.53
Rank	ops/succ	29295	30718	21169	31165	19838	14860	22907	20403	19260

Table 1: Algorithm IS-1 applied to  $X_{10}$

Climb probability		.6	.7	.8	.9	.95	.99
no	succ rate	.41	.47	.41	.21	.17	.02
Rank	ops/succ	19039	20339	21027	24297	23950	28505
with	succ rate	.50	.56	.46	.38	.29	.05
Rank	ops/succ	12299	18747	18811	24367	26558	31595

Table 2: Algorithm IS-2 applied to  $X_{10}$

Anneal probability		.1	.05	.01	.005	$Cost/ E(G) $
no	succ rate	0	.05	.25	.34	.19
ranking	ops/succ		23930	19854	25706	16520
with	succ rate	0	.07	.62	.53	.49
ranking	ops/succ		21872	19954	21418	20921

Table 3: Algorithm IS-3 applied to  $X_{10}$

our algorithms we find complete irregular weightings for  $X_5$  typically in under 7 seconds, while backtracking did not succeed in 122 hours.

All experiments reported in Tables 1-3 were carried out on an Apple Macintosh SE with a 68000 processor. As noted above, all operations are  $O(s)$ . Some approximate rates of execution (rounded to the nearest 5 operations per second) are given in table 4.

s		3	7	10	21	26	31	37
$O(s)$ ops/sec	IS-1	405	270	235	140	115	100	86
	IS-2	180	170	150	105	90	80	70
	IS-3	150	125	100	60	50	45	40

Table 4: Approximate rates of execution based on strength of weighting

These three algorithms behave similarly on other graphs when appropriate thresholds are chosen. For example, IS-1 applied to  $X_6$  required a threshold of 5,000 for similar results, and applied to  $X_{12}$  required a threshold of 250,000. On  $K_n$ , the use of Rank made a dramatic difference. For example, on  $K_{30}$ , algorithm IS-1 (with ClimbFactor of 16, ScanQuotient of 16, and threshold of 100,000) without Rank yielded a success rate of .1; while with Rank, its success rate was .61 and almost all successful weightings were found in fewer than 5,000 operations. As  $n$  grows, increasing

ClimbFactors and ScanQuotients appeared to improve the performance of IS-1 on  $K_n$ . Rank also yielded significant speed-up on the random graphs containing each edge with probability 0.5.

## 5 Conclusions

We have described several algorithms for irregularly weighting graphs. The algorithms are based on hill-climbing heuristics and are probabilistic in nature. As such, the algorithms are not guaranteed to terminate. However, we have shown the algorithms to be of great practical use. For example, our algorithms have generated irregular weightings on random graphs of about 1,000 edges, whereas backtracking becomes impractical for similar graphs of fewer than 100 edges.

In general, hill-climbing (and related techniques) excels over exhaustive search techniques when the search space is large and the density of solutions in the space is sufficiently great. The problem of irregularly weighting graphs is well suited to hill-climbing, as is shown experimentally by our results, and suggested analytically by the result in [EHLW2]. We believe these experimental techniques are broadly applicable. For example, [DS2] successfully applied these techniques to the problem of finding Room squares of side  $n$ . In that application, the search space is large,  $\mathcal{O}(n^{\binom{n}{2}})$ , but the solution space is also large,  $\mathcal{O}(e^{\binom{n}{2}})$ . [DS2]

The class of graphs on which we concentrated our efforts were the  $n \times n$  grids  $X_n$ . We have used our algorithms to irregularly weight all of the graphs in  $X_n$ , for  $2 \leq n \leq 23$ , with a strength of the known lower bound  $\lceil \lambda(X_n) \rceil$  ([GD]). Thus, we offer the following conjecture.

**Conjecture:**  $I(X_n) = \lceil \lambda(X_n) \rceil, n > 1$ .

In finding irregular weightings, lower thresholds can be used than those used here. In most of our experiments, the median number of operations for successful tries was less than half of the threshold. For example, one experiment using IS-1 on  $X_{10}$  had a success rate of .59 for 25,000 operations, and a success rate of .71 for 50,000 operations. Thus, in practice, a lower threshold can produce the first complete weighting in overall fewer operations.

We are continuing this research by expanding our repertoire of local search techniques, and attempting to better understand the factors that govern the performance of a given heuristic.

## References

- [CJLORS] G. Chartrand, M. Jacobson, J. Lehel, O.Oellermann, S. Ruiz, and F. Saba, Irregular networks, *Proc. 250th Anniv. Conf. on Graph Thry*, Fort Wayne, 1986.
- [CSS] L. Cammack, G. Schrag and R. Shelp, Irregularity strength of full d-ary trees, preprint.
- [DS1] J. Dinitz and D. Stinson, A fast algorithm for finding strong starters, *SIAM J. Alg. and Disc. Meth.*, **2** (1981), 50-56.
- [DS2] J. Dinitz and D. Stinson, On nonisomorphic Room squares, *Proc. AMS*, **89**:1 (1983).
- [DS3] J. Dinitz and D. Stinson, A hill-climbing algorithm for the construction of one-factorizations and Room squares, *SIAM J. Alg. and Disc. Meth.*, **8** (1987), 430-438.
- [EHLW1] G. Ebert, J. Hemmeter, F. Lazebnik, and A. Woldar, Irregularity strengths of certain graphs, preprint.
- [EHLW2] G. Ebert, J. Hemmeter, F. Lazebnik, and A. Woldar, On the number of irregular assignments on a graph, preprint.
- [FJKL] R. Faudree, M. Jacobson, L. Kinch, and J. Lehel, Irregularity of dense graphs, preprint.
- [FJLS] R. Faudree, M. Jacobson, J. Lehel, and R. Schelp, Irregular Networks, Regular Graphs, and Integer Matrices with Distinct Row and Column Sums, preprint.
- [GD] D. Garnick and J. Dinitz, *Empirical results of heuristic algorithms for the irregular weighting of graphs*, C.S. Rep. 89-1, Bowdoin College, Brunswick, ME., 1989.
- [G] A. Gyarfás, The irregularity strength of  $K_{m,m}$  is 4 for odd  $m$ , *Disc. Math.*
- [JL] M. Jacobson and J. Lehel, A bound for the strength of an irregular network, preprint.
- [KL1] B. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, *BSTJ*, **49**:2 (1970), 291-307.
- [KL2] L. Kinch and J. Lehel, The irregularity strength of  $tP_3$ , preprint.
- [KGV] S. Kirpatrick, C. Gelatt, and M. Vecchi, Optimization by simulated annealing, *Science*, **220**:4598 (1983), 671-680.
- [L] J. Lehel, Facts and quests on degree irregular assignments, *Proc. of 6th Intntl. Conf. on Graph Theory and Applications*, Kalamazoo, MI, 1988, to appear.
- [Li] S. Lin, Computer solutions of the traveling salesman problem, *BSTJ*, **44**:10 (1965).
- [MRRTT] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equation of state calculations by fast computing machines, *J. Chem. Phys.*, **21** (1953), 1087-1092.
- [PS] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, N.J., 1982, pp. 454-481.
- [RFRSK] B. Rothfarb, H. Frank, D. Rosenbaum, K. Steiglitz, and D. Kleitman, Optimal design of offshore natural-gas pipeline systems, *OR*, **18**:6 (1970), 992-1020.
- [S] D. Stinson, Hill-climbing algorithms for the construction of combinatorial designs, *Annals of Disc. Math.*, **26** (1985), 321-334.
- [SWK] K. Steiglitz, P. Weiner and D. Kleitman, The design of minimal cost survivable networks, *IEEE Trans. Cir. Theory*, **16**:4 (1969), 455-460.
- [W] L. Wille, The football pool problem for 6 matches: a new upper bound obtained by simulated annealing, *J. Combinatorial Theory (A)*, **45** (1987), 171-177.