Stochastic Local Search Algorithms for Minimizing Edge Crossings in Complete Rectilinear Graphs

Maxwell Young *maxwell_y@hotmail.com*

James Cook *jcook@cs.ubc.ca*

Ladan Mahabadi *LADAN12345@hotmail.com*

Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, BC
V6T 1Z4

Abstract

In this paper we consider the use of stochastic local search algorithms on the problem of obtaining upper bounds on rectilinear crossing numbers in complete graphs. For an integer K and a graph $G = (V, E)$, where $|V|$= n, the problem of answering whether or not $\overline{cr}(K_n)< k$ is known to be combinatorially difficult problem. [1, 4]. In response to this, we present two new algorithms that reduce the number of edge crossings in an initial drawing of G by employing local search heuristics. Furthermore, we provide analysis of both algorithms and offer insights into their performance. We conclude by providing upper bounds for K_n for n=39, 41, 42, 43, 44, 45, and $46 \leq n \leq 200$.

1 Introduction and Background

Stochastic Local Search (SLS) algorithms have demonstrated success in providing reasonable solutions to a variety of hard combinatorial problems [8]. SAT and Traveling Salesman Problem (TSP) are but two of the NP-complete problems to which stochastic local search algorithms have been successfully applied [8]. While SLS algorithms offer several high performance techniques for solving seemingly intractable problems, choosing the appropriate SLS techniques is dependent upon a thorough understanding of the properties of the problem and the characteristics of candidate SLS algorithms. The problem that we will be examining has presented a challenge to researchers for over the past four decades and it is our intent to ascertain whether or not certain SLS techniques can provide a viable method to meet this challenge. Let G=(V,E) be a graph drawn in the plane with vertex set V and edge set E. The crossing number of G, $cr(G)$, is the minimum number of edge crossings attainable over all such possible drawings of G. A closely related geometric problem involves adding the constraint that the edges of G be straight-line segments and that no three vertices are collinear. Such a drawing of G is called a rectilinear drawing and the rectilinear crossing number, denoted $\overline{cr}$(G) is the fewest number of edge crossings that can be achieved over all such drawings [4,5].

Garey and Johnson have shown the problem of ascertaining the crossing number of an arbitrary graph to be NP-complete [7]. For rectilinear graphs, the problem of determining whether $\overline{cr}(K_n)<k$ is not known to be NP-hard; however, it problem has shown significant resistance to various attempts at solving it. The search for rectilinear crossing numbers of the complete graph K_n began in the 1960s with the work of R. Guy. To date, the values for rectilinear crossing numbers of K_n are known only for $n \leq 12$. Many of the techniques used to enumerate the values for $\overline{cr}(K_n)$ require the use of computationally expensive combinatorial methods and such methods are insufficient in the face of larger values of n [16].

The problem of finding crossing numbers has found application in the area of Very Large Scale Integration (VLSI) circuit technology [5,9]. This technology is concerned with the construction of chips carrying large numbers of transistors. Reducing chip layout area and minimization of chip size is desirable because it allows for cheaper production costs and more reliable performance than larger chips [9]. However, another consideration is finding circuit layouts that minimize the number of circuit crossings and, hence, decrease the interference from capacitive coupling [9]. Striking this balance between chip size and circuit crossings is a crucial aspect of chip design and bears obvious economical importance in today's technology industry.

Our work on this problem has been aimed at applying the tools of a relatively new field of computer science, stochastic local search methods, to a relatively old mathematical problem. Below we provide some insight into the details of the problem domain with respect to gauging success and judging the value of new results.

In many instances of empirical assessment, there are many problem instances per problem size that can be examined. This aspect of a problem offers rich testing possibilities and, in this case, many established avenues of evaluation exist can be utilized. Unfortunately, in the context of crossing numbers for complete rectilinear graphs, there is precisely one problem instance per problem size. The uniqueness of this particular problem compel us to be flexible in our methods of evaluation. Many of the standard evaluation techniques are inadequate and unable to provide a meaningful measure of algorithm performance. To deal with this situation, we employ graphs of solution quality over time (SQTs) to give a graphical representation of how our algorithms are behaving. Another issue to address is the lack of prior work done in this area. Little work has been done on applying SLS techniques to the rectilinear crossing number problem. There have been attempts at using genetic algorithms to minimize crossing numbers, but these attempts have been somewhat limited in the size of n tackled [14,16]. Furthermore, to our knowledge, nobody has employed the type of SLS algorithms we propose in this paper. As a result, no data exists to which we might fairly compare the performance of our algorithms. While it would be beneficial to have some knowledge of prior results, the absence of such results is not an obstacle to the aims of this paper.

As is the always the case with employing SLS algorithms, our goal is to present algorithms that both perform quickly and find close-to-optimal, or even optimal, solutions. It is worth stressing that these two factors, solution quality and speed, are the two measures by which we can gauge the performance of our algorithms. Attaining 'good' upper bounds on the number of edge crossings for both known and unknown graphs constitutess, in itself, a certain amount of success. Additionally, the ability to achieve high quality, yet perhaps non-optimal, solutions is yet another indication of success. Ideally, one would like to be able to achieve both of these, often conflicting, goals; the two algorithms presented in this chapter go a long way to meeting these two challenges.

As previously mentioned, little relevant prior work involving similar SLS algorithms could be found. Furthermore, very little is known about optimal crossing numbers themselves. In fact, the optimal crossing numbers are known only for $n \leq 12$. Luckily, theoretical results exist for obtaining a lower bound on the crossing number. This formula is:

$$\overline{cr}(K_n) \geq \overline{cr}(K_{n-1}) \cdot \frac{n}{n-4}$$

Even more valuable are the results obtained by Aichholzer *et al* [1] who searched for crossing number values up to n=45; these were obtained by enumerating large numbers of inequivalent point sets of size n. Therefore, we have some information to which we can compare our candidate solutions. Finally, a number of papers have presented upper bounds for K_{81}. This graph, considered as a benchmark test, provides a useful measure by which to judge the performance of our algorithms.

2 Algorithm Description

In this section we present two stochastic local search algorithms that act to reduce the number of edge crossings in a graph $G = (V, E)$. In Section 2.1, an introduction is provided that illustrates the intricacies of implementing a program to solve the rectilinear edge crossing problem and some of the rationale behind the search heuristics of which both of our algorithms take advantage. The first algorithm, named Trio, consists of three different iterative local search algorithms acting in combination while the second implements a constructiive approach combined with random iterative improvement (RII). With Section 2.2 and 2.3, we offer a detailed outline of how both algorithms operate, respectively.

2.1 Implementation Issues

The most intuitive objective, and evaluation, function to employ is a measure of the number of edge crossings in a drawing of G. Clearly, by minimizing the evaluation function value, we improve our candidate solution. Enumerating the number of crossings in a graph requires an $O(n^3)$ calculation. While this cost becomes prohibitive as n grows in size, there are a few techniques that can be used to reduce the time of this calculation; the specifics of which are presented when we provide our implementation details. As is discussed later on, this cost actually grows to a $O(n^4)$ computation when used in one of our algorithms.

Another issue that deserves some attention is the method in which we represent the graph G. The most straightforward approach, and the one we utilize, is a grid system in which each vertex is assigned a unique point in the two-dimensional plane R^2. In adopting this representation, there are potentially two concerns that need to be addressed. The first pertains to the question of whether or not any bounds need to be set on the size of the x and y coordinates for the vertices. If we decide against setting such bounds, then we are faced with the possibility that the drawings of our graphs may be arbitrarily large. From a theoretical standpoint, this raises concerns regarding search stagnation. On the other hand, if we decide to place bounds on the size of the grid, then the values of these bounds must be chosen with some care. So far as we know, identifying the grid size that allows for the best chance of finding an optimal solution is an open problem and one with which we are not too concerned.

2.1.1 Vertex Responsibility

We will refer to the concept of vertex responsibility repeatedly in this paper. Consider a complete rectilinear graph K_n and a vertex of the graph v. As the name suggests, the vertex responsibility value of v is a measure of how the edges of v contribute to the overall number of edge crossings in the graph K_n. The responsibility value of v, resp(v), is calculated by examining each of the n-1 edges of v and summing up the number of edge crossings for each edge. Note that summing the responsibility values for all vertices in the graph K_n will usually *not* result in the total number of edge crossings but instead provide an overestimate since we are counting some crossings more than once.

2.1.2 Evaluation Function Cost

In the context of our problem, the evaluation function and objective function are identical and correspond to the number of edge crossings in the current candidate solution. Given the graph K_n, the computational cost of verifying the crossing number is $O(n^3)$ [15]. Even for moderate values of n, this cost is still prohibitive. Similarly, checking for collinearity in the graph K_n is also computationally expensive. We deal with both of these topics in more detail when we present the implementations of our algorithms. In any event, there is some encouraging evidence to show that this complexity is not necessarily fatal. For example, a previous experiment by Thorpe and Harris demonstrated that, in practice, the cost of the evaluation function is less costly than the worst case cost indicated by theory [15].

2.1.3 Heuristics

It is an open question as to whether or not the convex hull of any optimal drawing of a complete rectilinear graph is a triangle. Certainly, all drawings from n=3 to 12 (which has been proved optimal) display this characteristic. Furthermore, the best known drawings for values of n=13 to 45 (and n=81) follow this trend. This information is useful both in evaluating the solutions our algorithms discover and in the designing of an algorithm to provide optimal drawings of a complete rectilinear graph. Another attribute exhibited by optimal drawings is the tendency to spread out the total crossings evenly over all the vertices. All optimal and best known drawings of complete rectilinear graphs display approximately homogenous responsibility values for all n vertices. For example, with K_{10}, the responsibility values for an optimal drawing are: 27, 27, 27, 27, 25, 25, 25, 23, 23, 23. This lack of extreme variability in responsibility values seems to hint that distributing the number of crossings over all vertices leads towards better drawings of G.

2.1.4 Solution Density

The number of optimal drawings of K_n seem to vary depending on the value of n [1,4]. Results obtained by Aichholzer *et al* [1] suggest that odd values of n give rise to a greater number of possible optimal drawings relative to the number of optimal drawings that can be obtained for K_{n-1} or K_{n+1} when n is even. Given two optimal drwaings of K_n, one way to determine whether they differ is to compare the responsibilities for each vertex of each graph against one another. For instance, K_{10} has two non-isomorphic drawings as can be seen by comparing the responsibility values (where these values have been ordered from greates to least): 27, 27, 27, 27, 25, 25, 25, 23, 23, 23 versus 27, 27, 25, 25, 25, 25, 25, 25, 25, 23. Note that both sets of responsibility values add up to the same value (252) as should be expected.

The problem of determining the number of non-isomorphic drawings for K_n given an arbitrary n appears to be non-trivial [1,4]. While we have implemented a function to test for different drawings, we do not focus on this particular aspect of the rectilinear crossing number problem.

2.2 Implementation of Trio

As the name suggests, this algorithm employs three iterative local search algorithms with differing probability and conditions. Each of these three algorithms is significantly different from the others; however, there are some important similarities that require only a brief discussion. As mentioned previously, collinear vertices and degenerate vertex positions (vertices that share the same coordinates) should not be allowed by our algorithms. However, the cost of checking for collinearity and degenerate points after each search step in the algorithm is intolerably expensive. Fortunately, such checking is not required since by selecting a large enough grid, the chances that collinear or degenerate vertices arise is small. Only when each of the following three algorithms terminate is the candidate solution checked. If collinear or degenerate vertices exist then the solution is rejected and an error message is returned; otherwise, the candidate solution is kept.

One more point to keep in mind is that all of the algorithms presented in this paper are restricted to a 1-exchange neighbourhood. For any given search step, only one vertex is moved; clearly, an optimal solution can be attained by a sequence of such 1-exchange moves. Finally, it is interesting to point out that none of these three algorithms perform as well individually as they do in combination. The reasons for this and the details of how these three algorithms function are presented below.

2.2.1 Algorithm 1 - Greedy First Improvement Iterative Local Search (GFI)

As the name suggests, this algorithm takes a greedy approach to obtain good drawings of K_n. The vertex to be moved is selected on the basis of highest responsibility. This vertex is continually

```
procedure Greedy-First-Improvement-Iterative-Local-Search
    input: problem instance $\pi$ (ie. a complete rectilinear graph on n vertices), objective function $F$,
    optimal crossings $OPT$, maximum number of tries maxTries
    output: solution $s$ (ie. a complete rectilinear graph on n vertices with optimal number of edge crossings)
    $s$ := initialize($\pi$);
    current=$F(s)$;
    numTries :=0;
    while current>$OPT$  numTries$\leq$ $maxTries$ do
        $s'$ := move-worst-vertex($s$);
        temp= $F(s')$;
        if(temp<current)
            $s$ := $s'$;
            current=temp;
            calculateResponsibilities($s$);
            numTries++;
    end while
        if(degeneratePoints($s$) == false  collinearity($s$) == false)
            return $s$;
        else
            errorMessage();
end procedure Greedy-First-Improvement-Iterative-Local-Search.
```

Figure 1: Pseudocode for the Greedy First Improvement Iterative Local Search Algorithm (GFI).

tested in different positions in the plane at random until either a better graph is achieved or the set number of run steps is reached.

In terms of speed and the optimality of the obtained solution, this particular algorithm has demonstrated fairly good results for many of the smaller test cases. Intuitively, one of the reasons for its encouraging performance is that the algorithm, via its vertex selection process, tends to spread the number of crossings approximately evenly over all vertices. As discussed above, this is precisely one of the heuristics we would like our algorithm to employ.

Despite its encouraging performance, there are two drawbacks to GFI. The first is its tendency to get caught in local minima and stagnate for long periods of time. Ironically, this behaviour is the result of GFI's vertex selection method; a method which we have claimed performs well while the current candidate solution is still relatively far from the global optimum. However, upon closely approaching the global optimum, this method renders less promising results. At this point, it appears that the vertex with the highest responsibility is no longer the vertex that needs to be moved in order to achieve a better evaluation function value (this behaviour change of the problem provides the motivation our next algorithm). The second drawback of GFI is the high computational cost of running a search step. The cost of enumerating all the crossings in the graph K_n is O(n^3) due to the fact that not all vertices have to be examined in this procedure (described previously). However, in order to select the next vertex to move, the responsibilities of all the vertices have to be known. Unfortunately, this *does* seems to require the examination of *all* n vertices and so the total cost for each search step is O(n^4); other, more efficient methods, may exist and would certainly help improve the performance of this algorithm. Provided with a solution that is far from the global optimum, this cost is not very noticable since GFI operates on the basis of first improvement and, at this point, improvements are easy to find. This cost becomes highly prohibitive both when the algorithm nears the global optimum and begins to stagnate *and* when we test with large values of n. The pseudo-code for this algorithm is presented in Figure 1.

```
procedure Random-First-Improvement-Iterative-Local-Search
    input: problem instance $\pi$ (ie. a complete rectilinear graph on n vertices),
    objective function $F$, optimal crossings $OPT$
    output: solution $s$ (ie. a complete rectilinear graph on n vertices with optimal number of edge crossings)
    $s$ := initialize($\pi$);
    current=$F(s)$;
    numTries :=0;
    while current>$OPT$  numTries≤ $maxTries$ do
        $s'$ := move-random-vertex($s$);
        temp= $F(s')$;
        if(temp≤ $current$)
            $s$ := $s'$;
            current=temp;
            numTries++;
    end while
        if(degeneratePoints($s$) == false  collinearity($s$) == false)
            return $s$;
        else
            errorMessage();
end procedure Random-First-Improvement-Iterative-Local-Search.
```

Figure 2: Pseudocode for the Random First Improvement Iterative Local Search Algorithm (RFI).

2.2.2 Algorithm 2 - Random First Improvement Iterative Local Search (RFI)

This algorithm takes a more balanced approach to the vertex selection process. Rather than choosing the vertex with the greatest responsibility value, a vertex is selected at random. Again, this vertex is continually tested in different positions in the plane at random until either a better graph is achieved or the set number of run steps is reached.

In general, it does not achieve a reduction in the number of edge crossings with the same speed that algorithm 1 does. One of the reasons for this performance might be a result of the algorithm's tendency to sometimes attempt to move a vertex with a relatively low responsibility value. This runs counter to the observation that the best drawings of rectilinear graphs possess the quality that the responsibility values are spread almost evenly over all the vertices. Nevertheless, this algorithm possesses two very useful properties. First, the cost for a single search step is O(n^3), not O(n^4), since recalculating the responsibility values is not longer required. This results in a noticeable speedup in performance. Second, because RFI does not restrict itself to moving the vertex with highest responsibility, it demonstrates resistance to the problem of stagnation exhibited by GFI and allows the greedy algorithm to break out of local optima. The pseudo-code for this algorithm is presented in Figure 2.

Finally, another significant difference between GFI and RFI is the ability of latter algorithm to permit sideways moves. In the event that a vertex move results in neither a increase nor a decrease in the number of edge crossings, the new position is kept. This allows for potential rearrangement of all the vertices. It may be the case that the number of favourable positions to which a vertex can be moved is relatively small with respect the entire 1-exchange neighbourhood. Intuitively, this can occur when the favourable position to which to move a vertex lies is a very small face cut by the edges of the graph K_n. If none of the vertices are rearranged, this position can be consistently difficult to reach. By allowing RFI to shift the vertices around the plane, even in the absence of a resulting improvement, the chance of this undesirable possibility occurring is reduced.

```
procedure Bounded-Perturbation
    input: problem instance $\pi$ (ie. a complete rectilinear graph on n vertices), objective function $F$,
    number of worsening steps $W$, bound $B$
    output: solution $s$ (i.e.a complete rectilinear graph on n vertices with either more or less edge crossings)
    $s$ := initialize($\pi$);
    iterations :=0;
    $num - crossings$ := count($\pi$);
    while iterations$< W$ do
        $s'$ := move-random-vertex($s$);
        temp$= F(s')$;
        if(temp$-$current$\leq B$)
            $s$ := $s'$;
            current=temp;
    end while
        if(degeneratePoints($s$) == false  collinearity($s$) == false)
            return $s$;
        else
            errorMessage();
end procedure Bounded-Perturbation.
```

Figure 3: Pseudocode for the Bounded Perturbation Algorithm (BP).

2.2.3 Algorithm 3 - Bounded Perturbation

This algorithm was not designed to be used in isolation from the two former algorithms; the reasons for this will become obvious in a moment. First, we start with a graph G which has k edge crossings and we specify a constant B and a constant W. Like RFI, a vertex is selected at random and placed at another position in the plane. If the resulting graph G' has less than B edge crossings we accept G' as our new current solution. Otherwise, we shift the vertex to another position and re-evaluate in the same fashion. This process continues until we find an acceptable position or the set number of steps, W, is reached. Note that this algorithm does not forbid improving steps; however, as the global optimum is approached, it becomes more and more likely that the random vertex move implemented by BP will result in an increased number of crossings.

Out of all three, this algorithm, when operating by itself, displays the worst performance. However, it is very useful in allowing the two former algorithms to break out of local optima; this was its intended purpose and it seems to perform this task well. In general, this algorithm functions as a peturbing action and has provided good results when used in combination with the other two algoritms above as we shall see. The pseudo-code for this algorithm is presented in Figure 3.

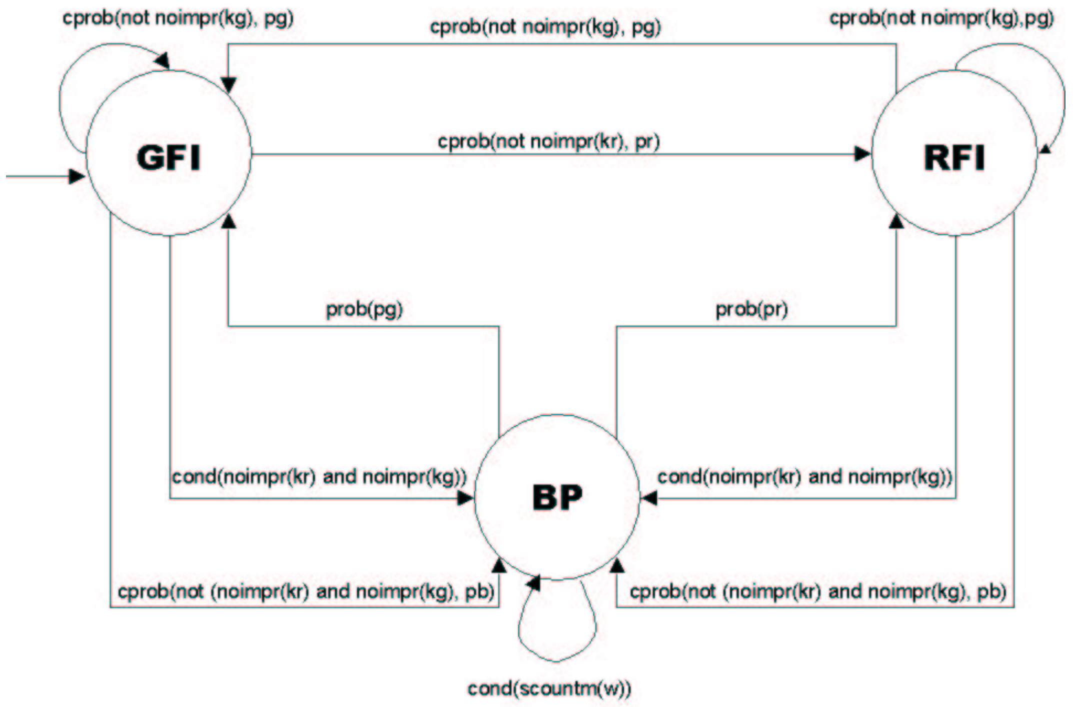
cprob(not noimpr(kg), pg)
cprob(not noimpr(kg), pg)
cprob(not noimpr(kg),pg)
GFI
cprob(not noimpr(kr), pr)
RFI
prob(pg)
prob(pr)
cond(noimpr(kr) and noimpr(kg))
BP
cond(noimpr(kr) and noimpr(kg))
cprob(not (noimpr(kr) and noimpr(kg), pb)
cprob(not (noimpr(kr) and noimpr(kg), pb)
cond(scountm(w))

Figure 4: GLSM for Trio

2.3 Trio

As discussed above, each of the three algorithms GFI, RFI, and BP have strengths and weaknesses. Trio is an attempt at using these three algorithms at different stages of problem solving process in order to capitilize on the successful behaviour of each and reduce any unfavourable performance.

Provided with a graph G, Trio begins by employing GFI to reduce the number of edge crossings. A positive constant value K_{GFI} is provided that provides an upper bound on the number of non-improving steps that are allowed before GFI is no longer allowed to be executed. At any point, based on a probability value P_{RFI}, Trio can switch from GFI to RFI. Again, a stagnation constant K_{RFI} is used to determine whether or not to keep utilizing RFI. With a probability of P_{GFI}, Trio can stop executing RFI and switch to GFI. This feature of being able to trade back and forth between GFI and RFI makes sense in the context of what was discussed above. By starting with GFI, Trio can very quickly approach the global optimum. Additionally, when GFI displays indications of stagnation, Trio can switch to RFI in order to further reduce the number of edge crossings in G. The final option available to Trio is the use of BP; there are two circumstances under which this occurs. First, when both GFI and RFI have reached their allotted stagnation values K_{GFI} and K_{RFI}, respectively, BP is executed. Second, if it is not the case that both GFI and RFI have reached their respective stagnation values, BP can still be executed with a probability P_{BP} (an emprical analysis of how P_{BP} affects the performance of Trio is provided later on in this paper). Regardless of how this transition occurs, BP is only run a prespecified number of times

W before Trio switches back to either GFI or RFI. Importantly, whenever BP is run, the recorded number of stagnating runs for both GFI and RFI are reset to zero. A GLSM for Trio is provided in Figure 4.

```
procedure Construction-with-RII(n,p,max_tries,$F$)
    input: graph size $n$, probability p, maximum number of moves max_tries, objective function $F$
    output: candidate solution $s$ (ie. a complete rectilinear graph on n vertices)
    $s$ = create_graph(3)
    while size($s$) $<$ $n$ do
        $s$ := add_one_vertex($s$)
        num_iterations := 0
        while num_iterations $<$ max_tries do
            $s'$ = move_newest_vertex($s$)
            if $F(s') < F(s)$
                $s := s'$
            end
            num_iterations := num_iterations + 1
        end while
    end while
    while not terminate($n$,$s$) do
        u := rand(0,1)
        if $u < p$ then
            s := Bounded-Perturb-Step($s$,max_tries,$F$)
        else
            s := Random-First-Improvement-Step($s$,max_tries,$F$)
        end
    end while
    if(degeneratePoints($s$) == false and collinearity($s$) == false)
    return $s$;
        else
            errorMessage();
end procedure Construction-with-RII.
```

Figure 5: Pseudocode for Construction with RII

2.4 Construction with Random Iterative Improvement (CRII)

This algorithm employs a different method in order to obtain good drawings of K_n. It is motivated by the widely believed hypothesis [1,4] that an optimal drawing of K_n can be attained by adding an extra vertex to the graph K_{n-1} and positioning it appropriately. A near optimal drawing of K_n can be created from a sequence of near optimal drawings of K_i, with $i = 1..n$. This near optimal drawing can provide a good starting point for a SLS algorithm to obtain an optimal drawing of K_n.

From the preceeding discusion, it is reasonable to believe that a constructive approach, which achieves a near optimal solution by adding only one vertex, can benefit from employing a random iterative first improvement strategy (RII). Indeed, the best results were obtained by combining RFI with BP to achieve a version of RII which could then be used to improve the solution achieved by the construction step.

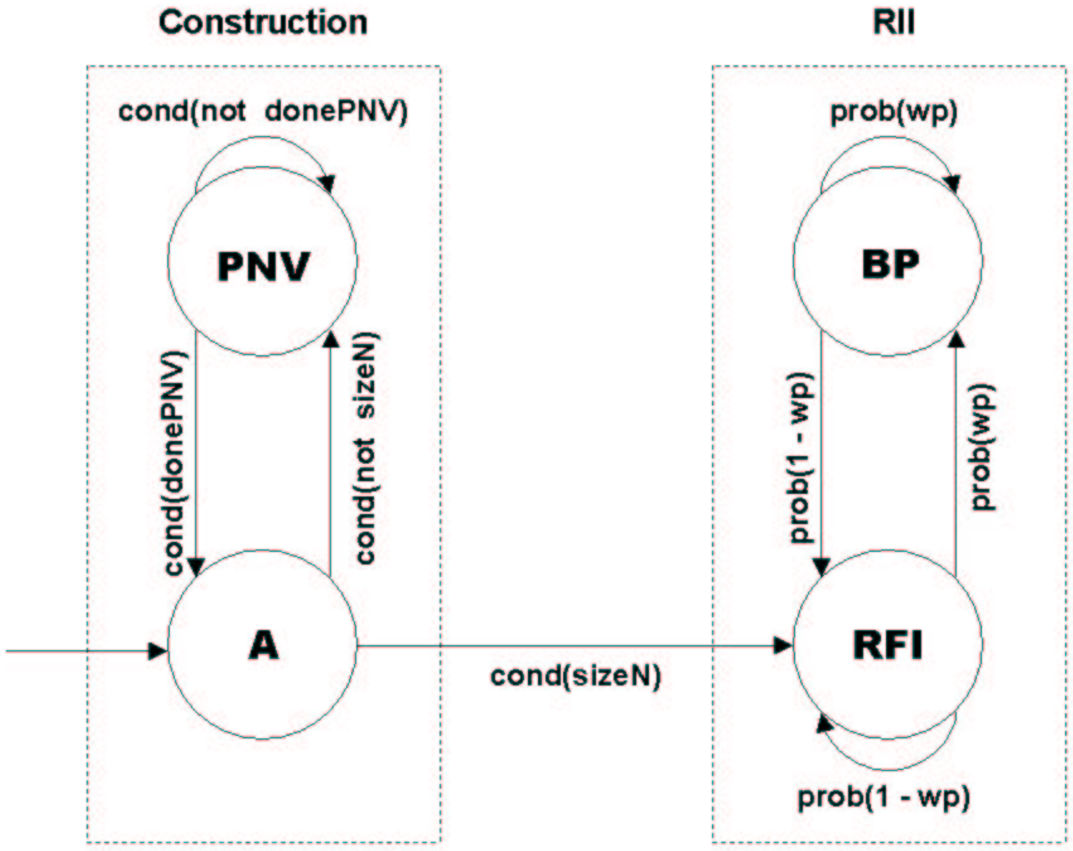

Figure 6: GLSM for Construction with RII

The construction algorithm starts with a drawing of K_3 (by providing the algorithm with three random non-colinear, non-degenerate, points). At each step, a vertex is added to K_i to obtain K_{i+1}.The new vertex added is randomly positioned in the plane and tested for δ positions. The final position of the vertex used is the position that adds the minimum number of edge crossings. The pseudo-code for this algorithm, abbreviated by CRII, is provided in Figure 5.

Calculating the crossings number for each vertex placement costs O(n^3). The only computation required for each position is calculating the number of edge crossings that are added by adding the new vertex. Once the position of the new vertex has been determined, the next step is performed and a new vertex is added. The construction phase terminates when K_n is reach for the specified n.

Once the desired graph size is reached, a local search phase is executed. The local seach step used is RII. A greedy improvement strategy, such as GFI, did not yield good results in testing. The constructed graph is close to optimal thus it is unlikely that any large improvements can be made. Consequently, the extra cost of computing the vertex with largest responsibility does not seem likely to offer any benefits over simply rearranging the added vertex. In Figure 6, a GLSM is also presented in order to provide an overview of CRII.

This algorithm provides a fast and efficient way to obtain relatively good upper bounds for the number of edge crossings in K_n for large values of n. Intuitively, this speed increase is traded for reduced solution quality since CRII does not search over all vertices in the graph but only on the added vertex. It was expected, and verified emprically later, that the solutions produced by this algorithm would be inferior to those produced by Trio. However, the speed increase is impressive and the results themselves provide a good starting place for more exhaustive algorithms.

3 Results

In this section we emprically analyze the perfomance of the two algorithms Trio and CRII. Section 3.1 provides the details of the tuning process for Trio with respect to the probability P_{BP} value for BP. Section 3.2 provides a small comparison between the behaviour of Trio versus RFI with respect to a moderately difficult problem instance. Section 3.3 is dedicated to analyzing the performance of Trio on complete rectilinear graphs for which previous good upper bounds are known. Completely new upper bounds for 39, 41, 42, 43, 44, 45, $46 \leq n \leq 65$, for n=81, and n=102 are given in Section 3.4 along with some new results involving previously undiscovered non-isomorphic drawings of K_n for n=25 and 26. CRII's performance on known graphs is provided in Section 3.5 and its results, including upper bounds for the number of edge crossings in K_n for values $4 \leq n \leq 200$, are presented in Section 3.6.

3.1 Tuning Trio

Trio has a few important parameters. Most of the values for these are easily established by simply doing a test run and viewing the results. For instance, by observing a few runs of Trio on a graph K_n, it is fairly easy to set the stagnation value K_{GFI}. One of the parameters that does require some tuning by more thorough emprical means is the probability, P_{BP}, of running BP.

3.1.1 The Effect of P_{BP}

In order to illustrate the effect of P_{BP} on performance, Trio was run 100 times with four different values for P_{BP} (and with all other parameters held constant) on relatively easy problem instance of K_{17}. The mean run-ties for each parameter value are presented down below in Table 1. The trend provided by the data was fairly obvious. The best values appear to have been obtained by setting P_{BP} to 0.001 whereas larger values resulted in extended run-times. This result follows the trend often displayed exhibited by SLS algorithms; a small perturbative or random walk step provides better performance. Using relatively large P_{BP} values leads to the algorithm performing perturbative steps long before reaching the the global optimum. Obviously, this is undesirable since progress made before the BP step is lost without any apparent compensation. By utilizing BP only infrequently, the algorithm has the opportunity to get closer to the global optima and take advantage of the occasional chance to escape from local minima. Although Table 1 does not provide definitive evidence, it supports the hypothesis that a relatively small value for P_{BP}

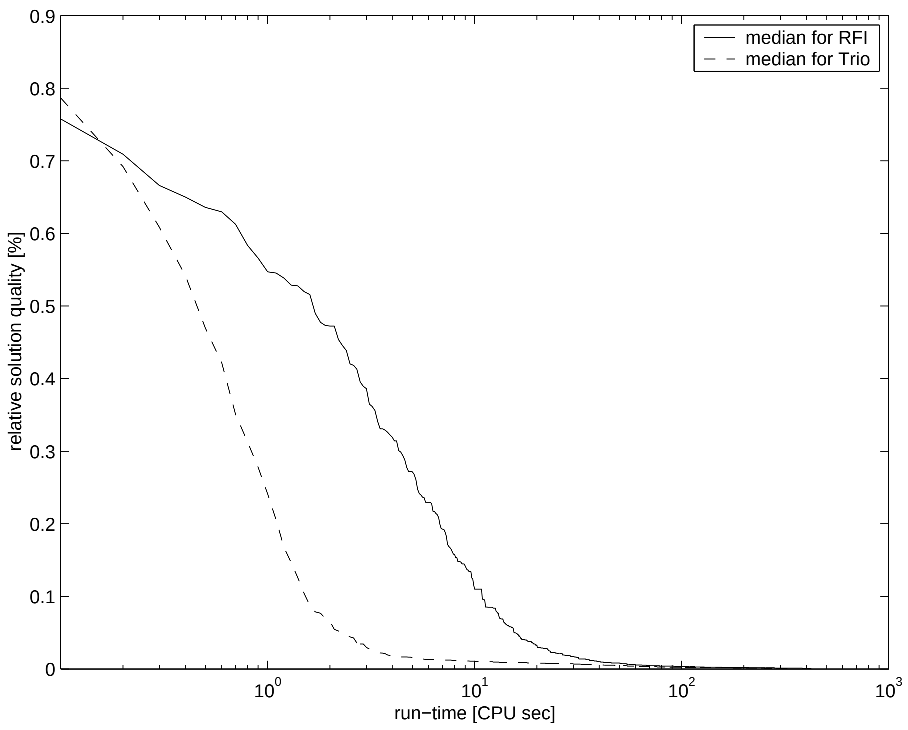

Figure 7: SQT of RFI and Trio running on problem instance K_{22}

provides the best results. Therefore, in our test runs, the probability for running PB was set to 0.003.

P_{bp}	Mean Run-Time [CPU sec]
15%	32.63
10%	30.58
5%	26.445
1%	28.82

Table 1: Mean Run-Times for Trio on K_{17} Using Different P_{BP} Settings

3.2 Performance of Trio over RFI

As an example of how much better Trio performs over any one of the single algorithms of GFI, RFI, and BP, we have provided the SQT in Figure 7 on the next page. This was obtained by taking 100 runs of Trio and 100 runs of RFI on the moderately difficult problem instance of K_{22}. There is a signicant difference between the two algorithms and it is quite obvious that Trio outperforms GFI. This type of comparison provides some measure of assurance that the motivation behind Trio is correct.

3.3 Performance of Trio on Problem Instances for which Upper Bounds Exist

Figures 8 to 12 are SQTs obtained from running Trio on the problem instance K_{17}, K_{19}, K_{22}, K_{25}, and K_{30}, respectively.

Due to the large amount of time required to run these experiments (a little over 27 CPU hours for K_{22}), only Figures 8 to 10 are based on data gathered on 300 runs. The SQTs for K_{25} and K_{30} are based on data gathered over 150 runs.

The behaviour of Trio is nicely illustrated by the above graphs. In each case, the initial graph was generated by randomly placing the appropriate number of vertices in the plane; not surprisingly, these initial graphs were consistently far from optimal. Starting from this initial placement of vertices, the SQTs of in Figures 8 to 12 demonstrate how the algorithm rapidly approaches the optimal value. The behaviour of Trio also changes as the value of n increases; a pronounced 'hump' can be observed at the beginning of the algorithms run which then transforms into a step descent towards the optimal value. Intuitively, we might expect to see such a trend which is most likely a consequence of the fact that the difficulty of the problem increases with the size of n. It should be noted that Trio confirmed many of the values found by Aichholzer *et al* [1] for values as high as n=30. In particular, it was able to verify the upper bound values for n=10, 11, 12, 13, 14, 15, 17, 20, 25, and 30 (other values were not attempted). Furthermore, Trio was able to improve upon the values given by Aichholzer *et al* [1] for n=39, 41, 42, 43, 44, 45,and 81.

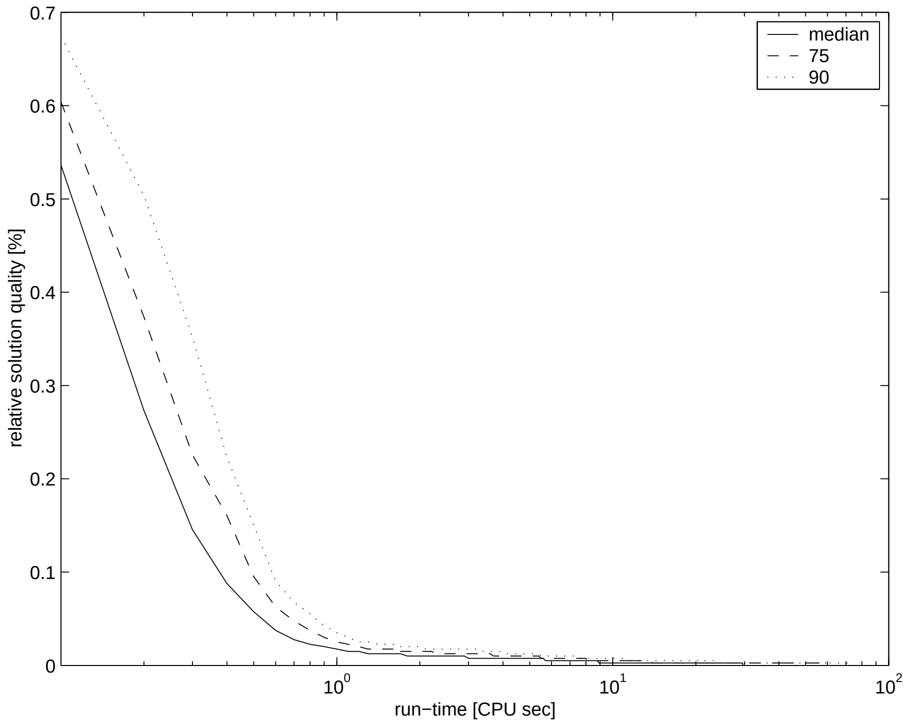

Figure 8: SQT for Trio running on K_{17}

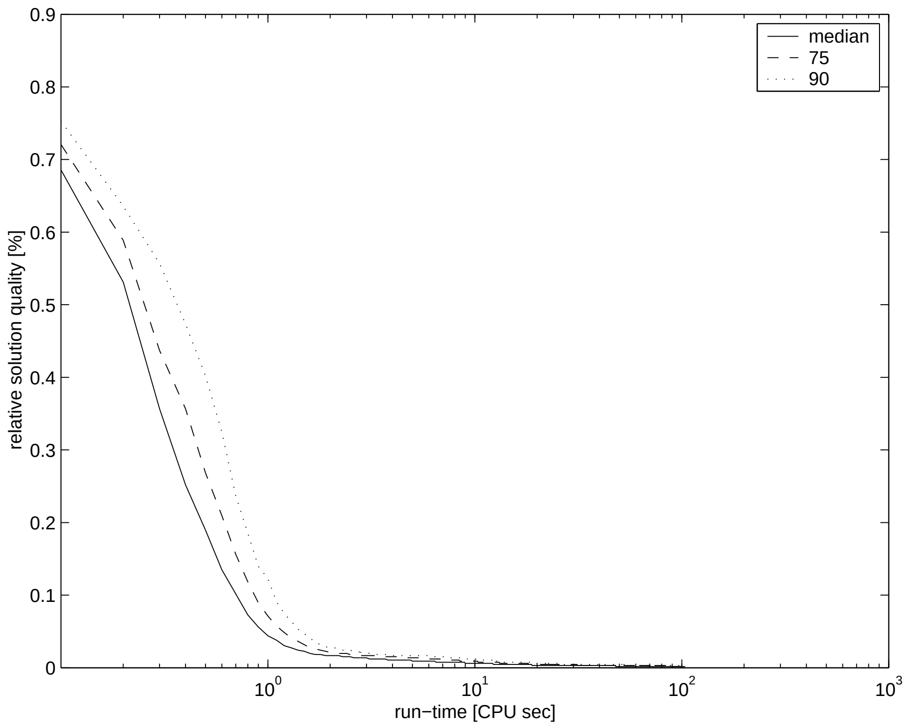

Figure 9: SQT for Trio running on K_{19}

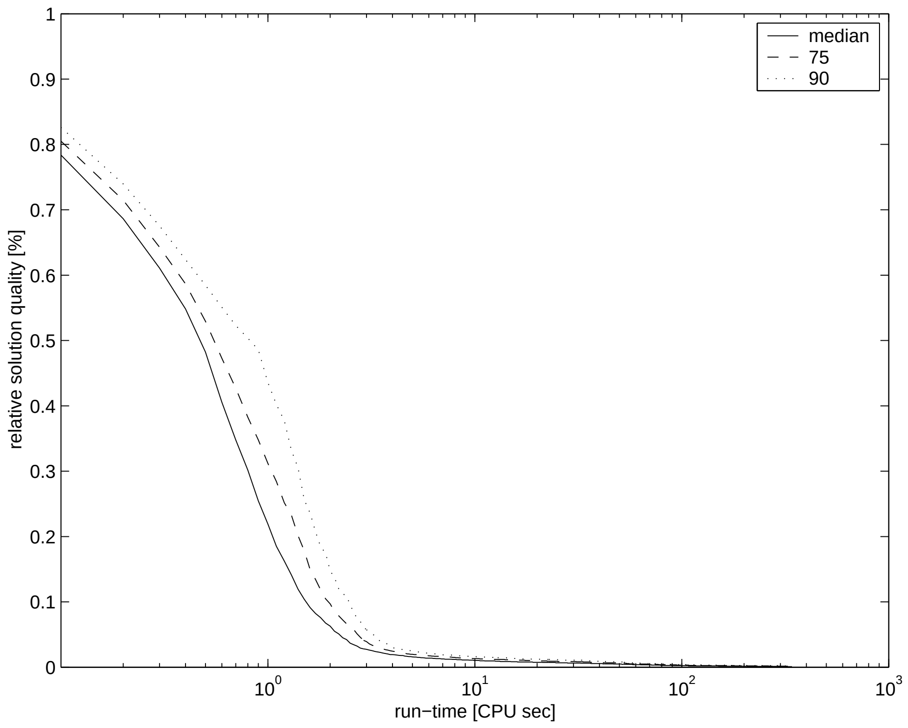

Figure 10: SQT for Trio running on K_{22}

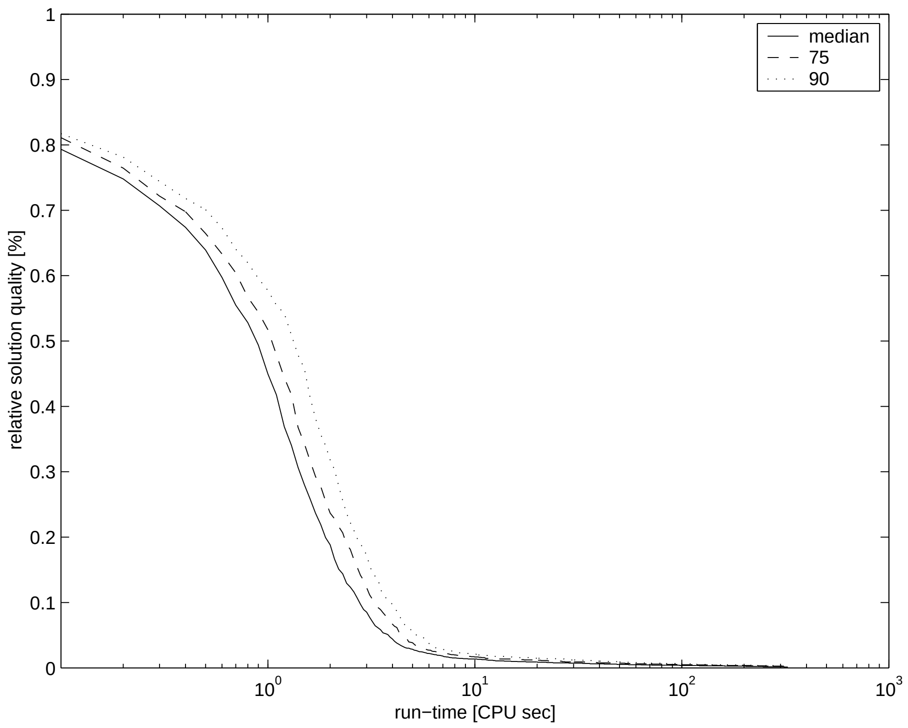

Figure 11: SQT for Trio running on K_{25}

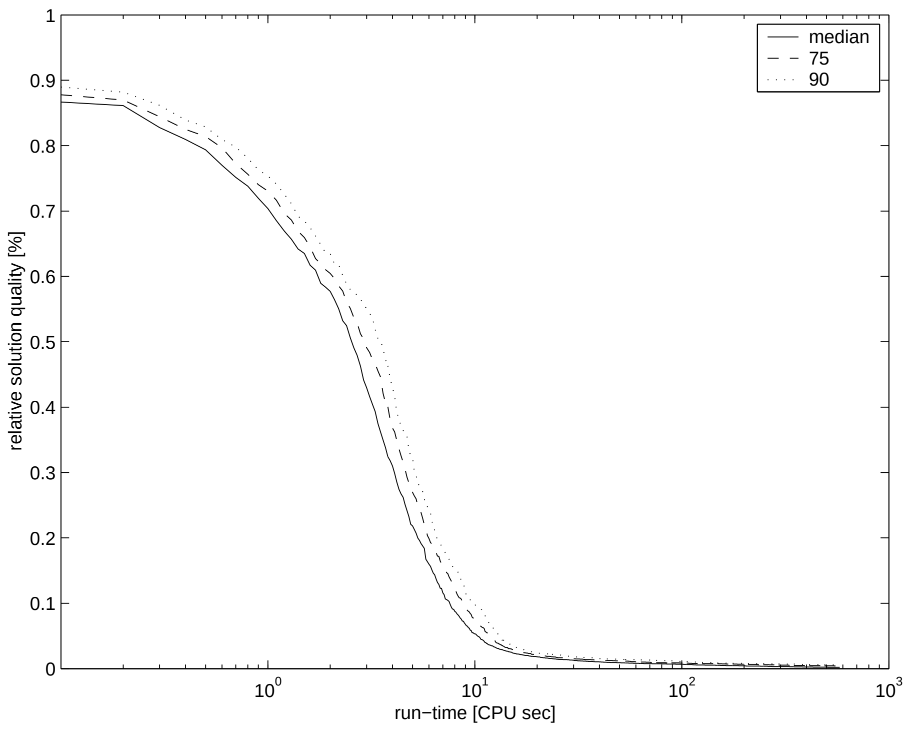

Figure 12: SQT for Trio running on K_{30}

For each value of n, the algorithm found at least one optimal drawing. However, as n increased, the ratio of successful runs (where an optimal drawing was obtained) to total number of runs decreased dramatically; this is reflected in Table 2.

Value of n	Number of Successful Runs	Total Number of Runs	Success Percentage
17	246	300	82%
19	56	300	18.6%
22	31	300	10.3%
25	5	150	3.3%
30	3	150	2%

Table 2: Performance of Trio on Various Problem Instances

3.4 Trio: New Upper Bounds and Non-Isomorphic Drawings

# n	Upper Bound	Lower Bound	Run-Time [CPU hours]
46	59611	50825	0.87
47	65153	55553	0.75
48	71166	60604	0.73
49	77576	65992	0.99
50	84421	71731	0.85
51	916684	77836	1.71
52	99322	84323	1.68
53	107573	91207	1.93
54	116278	98504	1.58
55	125515	106230	2.09
56	135332	110165	1.76
57	145806	118480	2.31
58	156470	127257	1.66
59	167984	136513	2.21
60	180048	146264	1.8
61	192831	156529	2.51
62	206112	167325	2.01
63	220101	178670	2.67
64	235198	190582	4.30
65	250688	203080	2.99
102	1594242	not calculated	2.81

Table 3: New Upper Bounds for $46 \leq n \leq 65$ and n=102

Above we presented upper bounds for $\overline{cr}(K_n)$ where $46 \leq n \leq 65$ and for n=102 along with the corresponding run-times required to achieve this solution.

The parameter settings were P_{BP}=0.03, W=1, B=5n, and the stagnation steps were provided by observing 2-3 runs; the grid size used was 1000000$\times$1000000. These experiments were carried out on a 1 GHz PIII processor under Linux.

Without prior results, it is hard to ascertain the quality of the results provided in Table 3. These lower bound values were calculated using the recursive formula provided at the beginning of this paper. It is important to keep in mind that the lower bound values are, with high probability, unatttainable; this is certainly true for the work done by Aichholzer *et al* [1] who found upper bounds for $11 \leq n \leq 45$. A plot has been provided in Figure 13 that demonstrates how well our values correspond with the trend set by Aichholzer *et al* [1]. There is a clear discrepancy, which increases as n increases, between the values obtained by Aichholzer *et al* and the theoretical lower bounds. The rate by which this discrepancy increases is relatively steady between both those values by Aichholzer *et al* and the values obtained by Trio for $46 \leq n \leq 65$. Based on the performance of Trio for values $11 \leq n \leq 45$ and the agreement between our results and those obtained by Aichholzer *et al* [1] (with respect to the theoretical lower bounds), we hypothesize that these results for $45 \leq n \leq 65$ are within 5% of the actual crossing number $\overline{cr}(K_n)$ for each respective n.

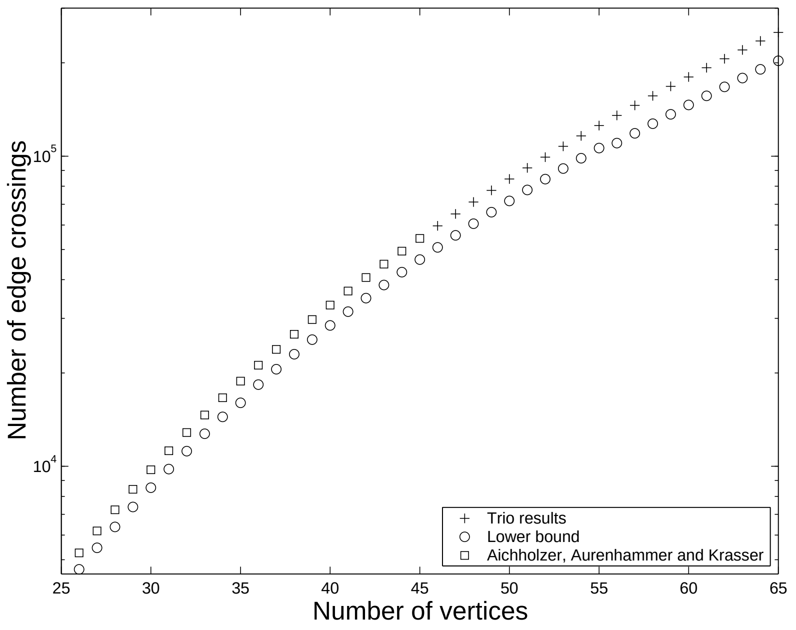

Figure 13: A plot of the theoretical lower bounds, the upper bounds set by Aichholzer, Aurenhammer, and Krasser, and the upper bounds found by Trio.

The last problem instance attempted by Trio is K_{81}. Historically, this particular problem instance has been used as a benchmark test. Hayward obtains a value of 659178, Jensen obtains a value of 630786 edge crossings, Singer's construction method provides a value of 625320 edge crossings, and the construction method of Brodsky *et al* obtains 623916 edge crossings [4]. Over the period of approximately 8.55 CPU hours, Trio managed to find a drawing with 619422 edge crossings; this is better than the previously mentioned values. Furthermore, after running the algorithm (a new run) for approximately 52.7 CPU hours, we achieved a drawing of K_{81} with only 618572 edge crossings which is better than the drawing obtained by Aichholzer *et al* [1] which had 618616 edge crossings via their construction method.

As a more recent development, Trio was run on the problem instances for n=39,41,42,43,44,and

45. This resulted in some new upper bounds which are presented below:

# vertices	Edge Crossings	Previous Upper Bound [1]	Time Required (CPU hours)
39	29729	29737	7.4
41	36730	36736	19.1
42	40633	40641	21.1
43	44862	44872	13.2
44	49389	49397	11.8
45	54261	54285	6.0

Table 4: New Upper Bounds with Trio

Finally, as one further new result, we present two non-isomorphic drawings of K_{26}. Previous to this work, only one non-isomorphic drawing for this problem instance was known. Unfortunately, the time constraints placed on this project limited the amount of work we could do on the problem of enumerating the number of non-isomorphic K_n for various values of n. Additionally, we also found an additional non-isomorphic drawing of K_{25} for which only three such drawings were previously known [1].

3.5 Performance of CRII on Known Problem Instances

The parameters for the RII algorithm and construction are given in Table 5. The cutoff is the maximum number of iterations performed by the RII algorithm. For every instance of K_n, the probability of executing a bounded perturbation step is 0.02. Furthermore, the bound on number of edge crossings added by a perturbation step is infinity.

# vertices	Grid Size	Cutoff
17	1000$\times$1000	2000
19	3000$\times$3000	3000
22	5000$\times$5000	3000
25	7000$\times$7000	3000

Table 5: Parameters for construction with RII

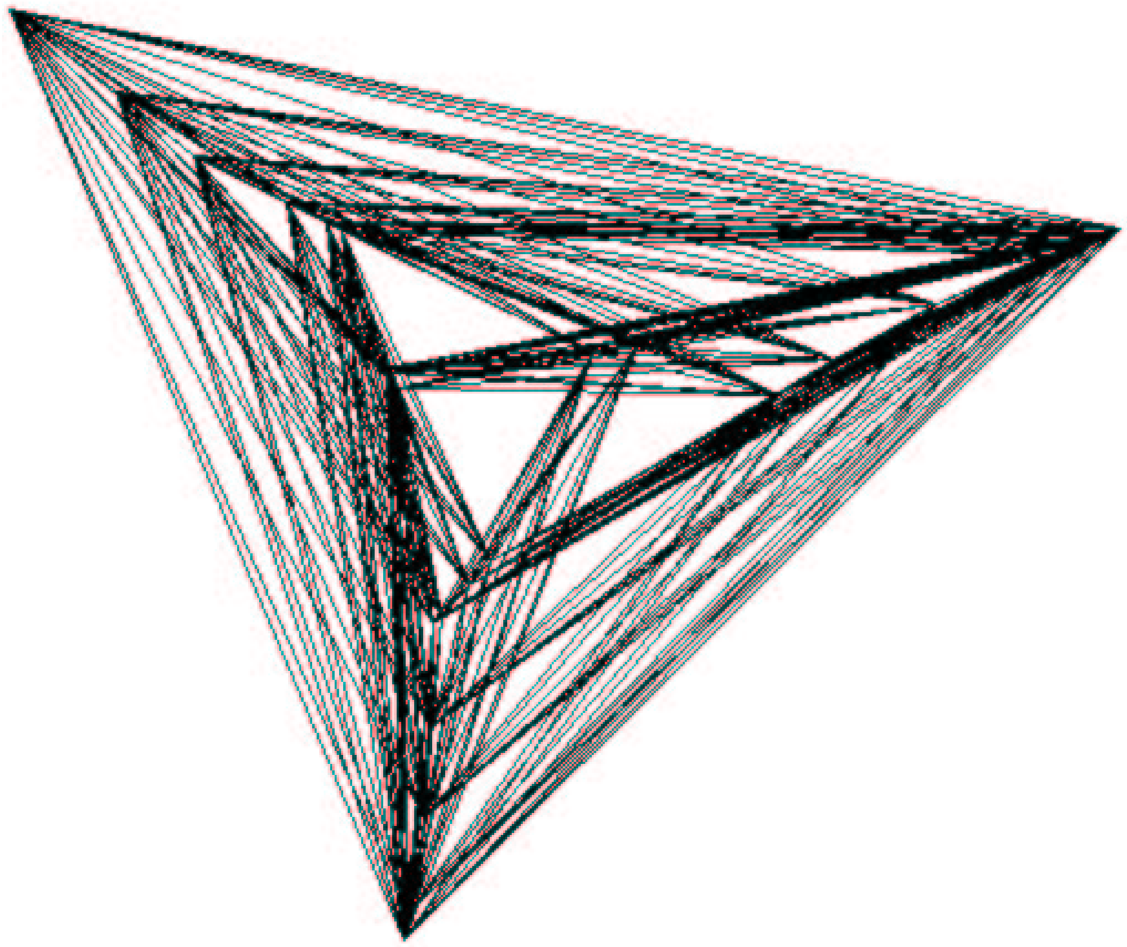

Figure 14: An Optimal Drawing of K_{26}

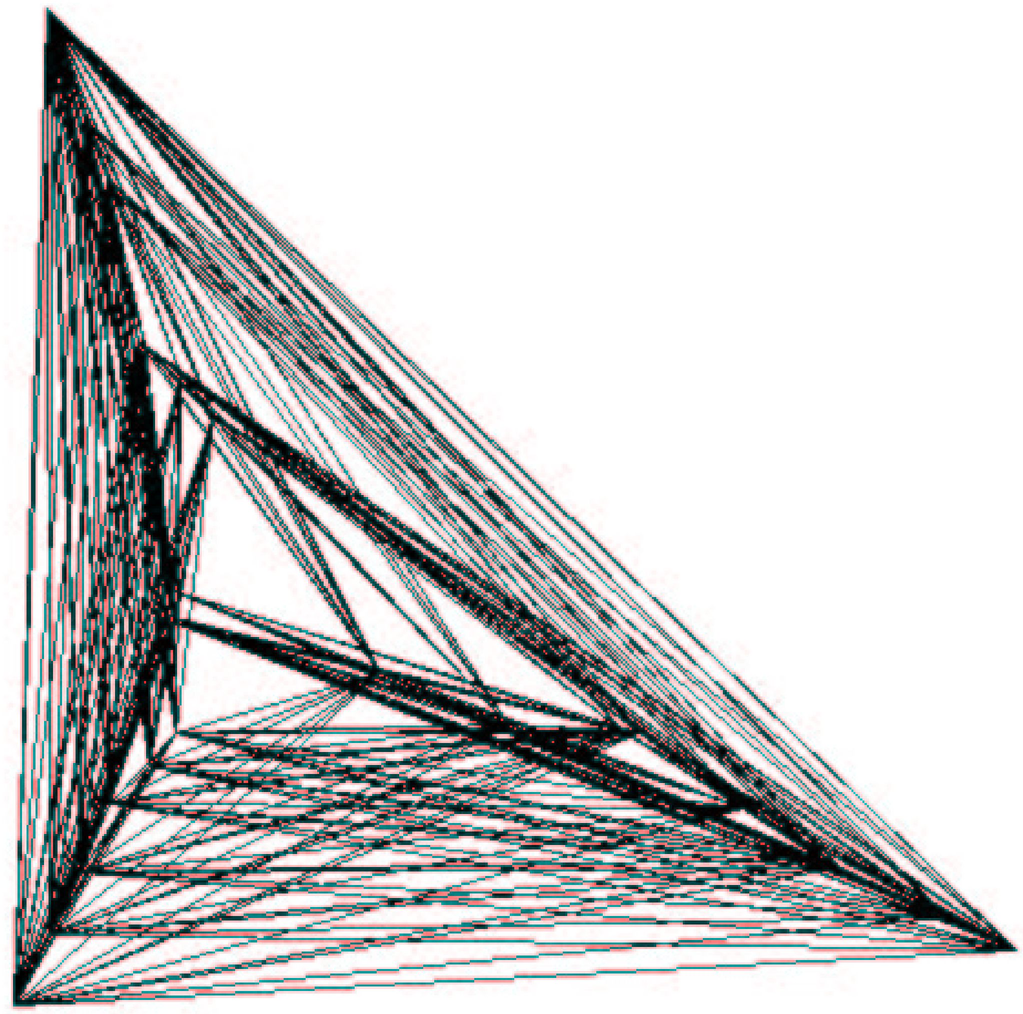

Figure 15: Another Optimal Drawing of K_{26}, non-isomorphic to the one in Figure 14.

The construction method provides a good starting point for the SLS algorithms. However, the simple RII algorithm has difficulty improving the crossing number of the constructed graph. The construction method might be building solutions that lie close to local minima and yet still be many steps away from the optimal solution. The initial results seem to verify this fact; however, more testing needs to be done. The simple RII algorithm may not be effective enough to improve the crossing number of the constructed graph. Better results may be obtained by using another SLS algorithm on the constructed graphs.

The construction algorithm yields crossing numbers that are close to optimal in a very short time. The resulting edge crossings can be used as upper bounds for the crossing numbers of K_n for very large graphs which have no known upper bounds. From our experiments, the relative solution quality from using only construction is less than 0.21% (for each instance of K_{10} to K_{45}). The median solution quality of contruction is less than 0.06% (for each instance of K_{10} to K_{45}). The time to achieve these results using construction is much smaller than using the SLS algorithms.

It is interesting to note that the construction method was able to reach the known upper bound for K_{17} yet was unable to reach the known upper bound for K_{16}. This means that no run of the algorithm reached the lower bound of K_{16} but at least one run was able to create a drawing of K_{17} with crossing number equal to the lower bound from K_{16} with crossing number greater than the lower bound. This might imply that a good drawing of K_n is not needed to get a good drawing of K_{n+1} by adding a single vertex to K_n.

Figures 16 to 19 provide SQTs of CRII's performance on the problem instances K_{17}, K_{19}, K_{22}, and K_{25}, respectively. These SQTs were obtained from the same data that was used to provide Table 5. Like Trio, CRII exhibits some of the same rapid descent towards the optimal value; however, this behaviour is far less pronounced as is evidenced by the lower quality of the solutions it provides.

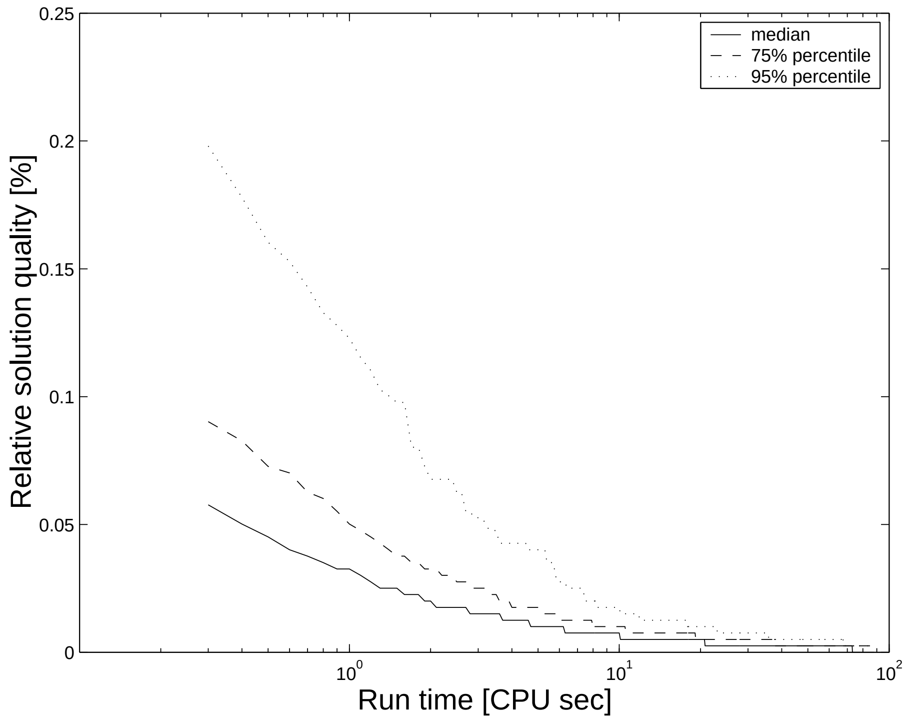

Figure 16: SQT for Construction with RII for K_{17}

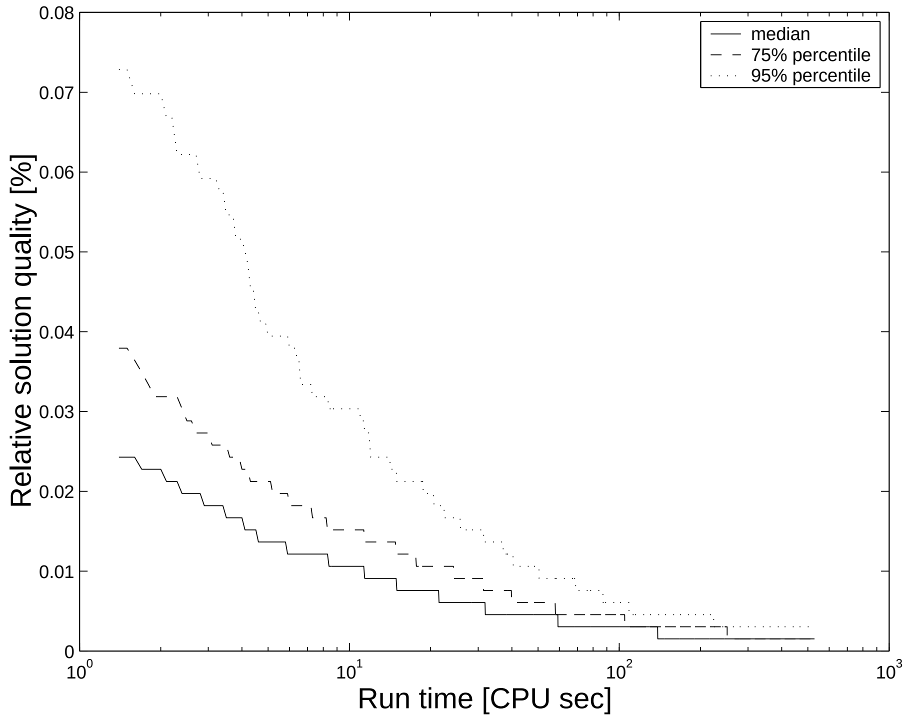

Figure 17: SQT for Construction with RII for K_{19}

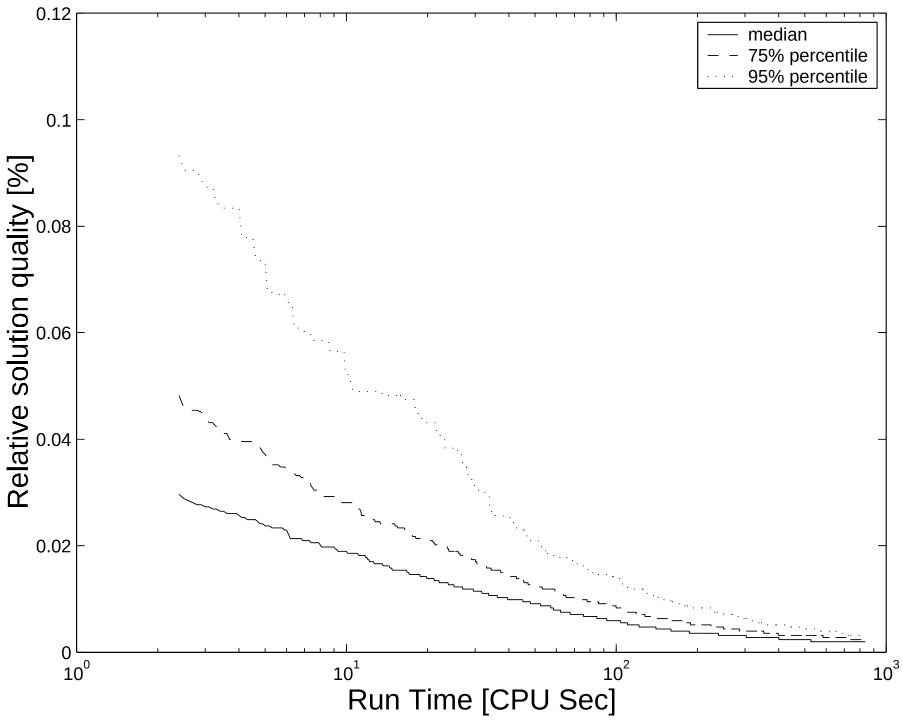

Figure 18: SQT for Construction with RII for K_{22}

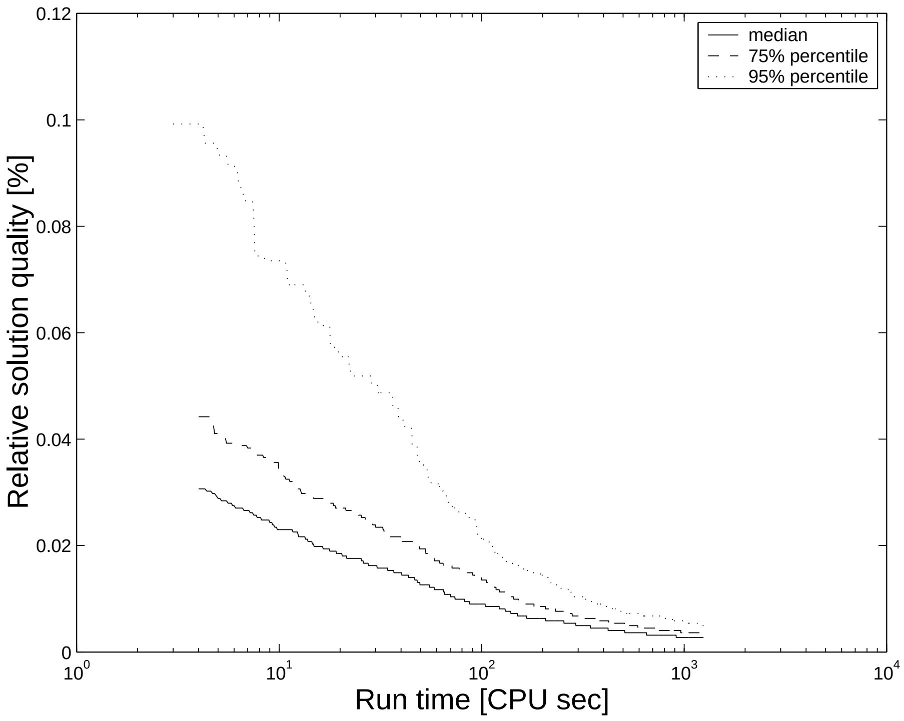

Figure 19: SQT for Construction with RII for K_{25}

3.6 CRII: New Upper Bounds

Table 6 contains statistics for the construction method. In each case, 300 runs of the construction method were performed by constructing a graph from 3 vertices to 81 vertices. At each construction step, the new vertex is tried at 300 positions.

Table 6: **Construction Statistics**

Number of vertices	Upper Bound*	Minimum constructed	Median of constructed	Maximum constructed	Standard deviation	Average time [CPU seconds]
4	0	0	0	0	0	< 0.001
5	1	1	1	1	0	< 0.001
6	3	3	3	4	0.237	< 0.001
7	9	9	9	11	0.199	0.010
8	19	19	19	24	0.550	0.020
9	36	36	36	44	0.944	0.04
10	62	62	63	75	1.465	0.060
11	102	102	102	122	1.969	0.099
12	153	153	157	180	3.173	0.141
13	229	229	231	263	4.352	0.208
14	324	324	330	379	5.776	0.289
15	447	447	457	521	7.927	0.391
16	603	605	616	700	10.608	0.519
17	798	798	814	922	13.585	0.679
18	1030	1033	1055	1175	17.852	0.869
19	1318	1322	1348	1520	23.191	1.099
20	1658	1664	1696	1919	29.249	1.368
21	2057	2073	2110	2373	36.177	1.686
22	2530	2550	2593	2921	43.691	2.057
23	3079	3101	3153	3529	51.712	2.488
24	3704	3731	3802	4256	62.306	2.980
25	4434	4466	4545	5110	74.550	3.544
26	5256	5300	5396	6016	87.216	4.179
27	6186	6230	6359	7068	103.547	4.902
28	7244	7309	7444	8326	121.410	5.710
29	8427	8497	8664	9761	142.704	6.616
30	9745	9852	10026	11213	161.046	7.623
31	11221	11353	11535	12891	187.498	8.748
32	12846	12976	13217	14772	217.333	9.986
33	14642	14816	15074	16872	246.500	11.357
34	16632	16842	17124	19193	279.197	12.859
35	18820	19046	19371	21516	311.688	14.506
36	21191	21477	21828	24052	346.734	16.299
37	23817	24129	24519	26901	394.785	18.263
38	26660	26986	27441	30199	446.600	20.397

Table 6: **Construction Statistics**

39	29737	30197	30637	33697	493.483	22.721
40	33093	33557	34096	37455	545.294	25.221
41	36736	37274	37856	41620	604.742	27.939
42	40641	41232	41884	46107	663.879	30.857
43	44872	45530	46245	50954	738.776	34.010
44	49397	50179	50922	55986	811.864	37.381
45	54285	55143	55984	61695	903.932	41.012
46	-	60424	61402	67531	995.045	44.891
47	-	66183	67183	73983	1087.331	49.064
48	-	72289	73364	80863	1191.392	53.485
49	-	78756	80011	87810	1297.189	58.217
50	-	85757	87034	95272	1401.879	63.241
51	-	93182	94533	103540	1520.458	68.620
52	-	101024	102500	111948	1627.455	74.293
53	-	109307	111002	121299	1768.597	80.333
54	-	118161	119992	130947	1883.350	86.730
55	-	127553	129515	141069	2022.124	93.521
56	-	137421	139558	152148	2182.838	100.658
57	-	147950	150197	163814	2350.276	108.248
58	-	159096	161487	176132	2536.895	116.217
59	-	170814	173394	189340	2747.645	124.651
60	-	183245	185968	203117	2932.244	133.506
61	-	196053	199081	218307	3160.374	142.854
62	-	209758	212920	233491	3398.250	152.669
63	-	224117	227501	250109	3631.435	163.022
64	-	239259	242894	267939	3891.513	173.831
65	-	255034	258927	286264	4155.529	185.208
66	-	271681	275848	305145	4448.870	197.098
67	-	289040	293550	326482	4792.995	209.608
68	-	307220	312054	344823	5047.793	222.616
69	-	326477	331363	366589	5329.289	236.312
70	-	346129	351702	389060	5666.287	250.564
71	-	367043	372856	411833	6015.190	265.530
72	-	388922	394902	436057	6368.238	281.061
73	-	411456	418054	462094	6710.115	297.339
74	-	435383	442158	488642	7040.832	314.256
75	-	460134	467267	517234	7417.673	331.978
76	-	485914	493539	547778	7895.387	350.328
77	-	512865	520678	577881	8279.141	369.501
78	-	540832	549126	607109	8691.741	389.384
79	-	569713	578482	640391	9123.466	410.149
80	-	600289	609022	675090	9655.961	431.621
81	618616	631304	640908	713076	10247.956	454.039

For the values obatined for n >81, four runs were performed by constructing a graph from 3 vertices to 200 vertices. The best values are given in Table 7.

# vertices	Best crossing number	# vertices	Best crossing number	# vertices	Best crossing number
82	670865	122	3401273	162	10762812
83	705068	123	3515302	163	11037850
84	741202	124	3632070	164	11316009
85	777765	125	3753597	165	11600913
86	816013	126	3876778	166	11888322
87	855943	127	4004537	167	12172961
88	896815	128	4133111	168	12468500
89	940026	129	4266236	169	12767378
90	985315	130	4402602	170	13073323
91	1029828	131	4542686	171	13382490
92	1077491	132	4686747	172	13704484
93	1126803	133	4831919	173	14028723
94	1176528	134	4980441	174	14356709
95	1228557	135	5136597	175	14693853
96	1281911	136	5294937	176	15037795
97	1337186	137	5455826	177	15385068
98	1395561	138	5619583	178	15741130
99	1455432	139	5783660	179	16101320
100	1517363	140	5955041	180	16469293
101	1581197	141	6130759	181	16843241
102	1645699	142	6309342	182	17221348
103	1712941	143	6492899	183	17609355
104	1781323	144	6681468	184	18003575
105	1850824	145	6873504	185	18407920
106	1922745	146	7067191	186	18812087
107	1996658	147	7261392	187	19221690
108	2073209	148	7461474	188	19644043
109	2152361	149	7666035	189	20069373
110	2233540	150	7877120	190	20504571
111	2317617	151	8092305	191	20943235
112	2401173	152	8308386	192	21382504
113	2489646	153	8534430	193	21836608
114	2581183	154	8760349	194	22304522
115	2674178	155	8994790	195	22779910
116	2769897	156	9232963	196	23251943

# vertices	Best crossing number	# vertices	Best crossing number	# vertices	Best crossing number
117	2867979	157	9477577	197	23738649
118	2970345	158	9726120	198	24237351
119	3073165	159	9977175	199	24731019
120	3179186	160	10234110	200	25235896
121	3288862	161	10493346		

Table 7: Construction Statistics for Large Rectilinear Graphs

The above experiments with CRII were run using a 1 GHz PIII processor under Linux.

4 Conclusions and Future Work

Historically, providing ’good’ drawings of complete rectilinear graphs has allowed for much of the progress towards obtaining values for $\overline{cr}(K_n)$ [4]. Determining whether or not a particular drawing is ’good’ is a difficult and qualitative task; however, a comparison with previous results and theoretical lower bounds can usually provide enough information with which to judge the quality of a particular drawing. Our work here has provided a number of new upper bounds which, in the light of all available evidence, seem to be of high quality. That being said, we do not doubt that, as the quest for new rectilinear crossings number proceeds, these upper bounds will be replaced by better (and perhaps even optimal) values. In particular, the methods used by Aichholzer *et al* [1] promise to uncover better upper bounds if such an endeavour is ever undertaken.

One of the most encouraging results of our work on this problem is the success with which our two rudimentary SLS algorithms achieved both in terms of speed and solution quality. Trio was remarkably successful in providing high quality drawings. The main drawback to Trio’s implementation is the slowdown in speed that results when larger and larger values of n are used. Conversely, CRII was able to output solutions of relatively good quality (although, not as good as those provided by Trio) at very high speeds.

In summary, the application of stochastic local search techniques to rectilinear crossing number problem seems very promising as is demonstrated through the analysis above. Indeed, future research opportunities exist with respect to this problem. In terms of the work done in this paper, there are a couple of ideas that deserve some attention:

1) Combining Trio with CRII might produce an algorithm that inherits both Trio’s ability to find high quality solutions and some of CRII’s performance speed.

2) CRII’s ability to provide high quality solutions might be improved by allowing it to move more than just the added vertex at any given construction step.

A more advanced method that might deserve future attention is based on the idea of using the points of intersection to find optimal vertex placements. A few of these ideas that have been developed by the authors of this paper are:

1) Perturbation around the neighborhood of an intersection point:
In case of an intersection, one could move one of the vertices of the graph closer to the intersection point incident on its edge. By perturbing around the intersection point, the orientation of this vertex in regards to other vertices changes and consequently affects the total number of crossings in the graph. It seems logical and promising to choose the worst vertex of the graph, the vertex with the highest number of edge crossings, to be perturbed around the cluster of intersections of edges.

2) Displacement of the worst vertex closer to the cluster of intersections:
In this technique, one would move the worst vertex of the graph as defined above closer to the intersection cluster of the graph. In particular, one could sample the intersection cluster for intersection points and then move the worst vertex to either the mean of the sampled intersection points or to their median.

3) A Combination technique:
As preliminary testing shows, a combination of the above two techniques for minimizing the crossing number of a rectilinear complete graph is, indeed, very successful. One could randomly choose between the above two proposed methods in order to minimize the number of crossings.

Acknowledgements. We gratefully acknowledge Holger Hoos for all of his idea inspiring insights, Will Evans for his helpful discussion and initial ideas on this project, and Steph Durocher for his encouragement and for sharing his extensive knowledge of the crossing number problem with us. We are also indebted to Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser; their work on rectilinear crossing number problem was invaluable. Maxwell would also like to thank Mirela Andronescu for fixing his glasses (so that he was not completely blind and could work on this project) and for offering Latex formatting help. Finally, Maxwell would also like to thank Tanja Radeka and Ali Taleghani for encouragement and challenging us to come up with new results.

References

[1] O.Aichholzer, F.Aurenhammer, and H.Krasser. On the Crossing Number of Complete Graphs. Published electronically at http://www.cis.tugraz.at/igi/- oaich/publications.html, 2002.

[2] D.Bienstock and N.Dean. Bounds for Rectilinear Crossing Numbers. *Journal of Graph Theory*, 17:333-348, 1993.

[3] D. Bienstock. Some Provably Hard Crossing Number Problems. Published electronically at http://portal.acm.org/citation.cfm?id=98581coll=portal- dl=ACMret=1FullText.

[4] A. Brodsky, S. Durocher, E. Gethner. Toward the Rectilinear Crossing Number of K_n: New Drawings, Upper Bounds, and Asymptotics. Published electronically at http://www.cs.ubc.ca/ durocher/research/research.html.

[5] A.Brodsky, S.Durocher, and Ellen Gethner. The Rectilinear Crossing Number of K_{10} is 62. Published electronically at http://www.cs.ubc.ca/ durocher/ research/research.html, 2001.

[6] N.Dean. Mathematical Programming Formulation of Rectilinear Crossing Minimization. Published electronically at http://www.caam.rice.edu/ nated/- publics/allpapers.html, 2002.

[7] M.R.Garey and D.S.Johnson. Crossing Number is NP-Complete. *SIAM Journal of Algebraic and Discrete Methods*, 4:312-316, 1983.

[8] H.H.Hoos and T.Stützle. *Stochastic Local Search- Foundations and Applications*. Morgan Kaufmann Publishers to appear in 2003.

[9] F.T.Leighton. New Lower Bound Techniques for VLSI. *Mathematical Systems Theory*, 17:47-70, 1984.

[10] O.Martin, S.Otto, and E.W.Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, June 1991.

[11] J.Pach. Crossing Numbers. Published electronically at http://www.cs.gc.cuny.- edu/ pach/, 2000.

[12] J.Pach, J.Spencer, and G.Toth. New Bounds on Crossing Numbers. *15th ACM Symposium on Computational Geometry*, 124-133, 1999.

[13] R.B.Richter and C.Thomassen. Relations Between Crossing Numbers of Complete and Complete Bipartite Graphs. *American Mathematical Monthly*, 104(2):131-137, Feb 1997.

[14] A. Rosete-Suarez, A. Rodriguez, and M. Sebag. Automatic Graph Drawing and Stochastic Hill Climbing Published electronically at http://citeseer.nj.nec.com/cache/papers/cs/20281/http:zSzzSzwww.lri.frzSz rosetezSzgechcgd.-pdf/automatic-graph-drawing-and.pdf

[15] B.Selman, H.Levesque, and D.Mitchell. A New Method for Solving Hard Satisfiability Problems. *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440-446, July 1992.

[16] J.T.Thorpe and F.C.Harris. A Parallel Stochastic Optimization Algorithm for Finding Mappings of Rectilinear Minimal Crossing Problem. Published electronically at http://www.cs.unr.edu/ fredh/papers/journal/Journal.html, 1996.

[17] U. Wagner. On the Rectilinear Crossing Number of Complete Graphs. Published electronically at http://www.inf.ethz.ch/ uli/ , 2003.