Lightweight Arc-Consistency Algorithms[1]

M.R.C. van Dongen
dongen@cs.ucc.ie

Cork Constraint Computation Centre
CS Department
University College Cork
Western Road
Cork
Ireland

Technical Report TR-01-2003

January 2003

Available from http://csweb.ucc.ie/~dongen/papers/4C/03/4C-01-2003.pdf

[1]Please save some trees and do not print the appendix of this report.

Abstract

Arc-consistency algorithms are the workhorse of many backtrack algorithms. Most research on arc-consistency algorithms is focusing on the design of algorithms that are optimal when it comes to worst case scenarios. This report will provide experimental evidence that, despite common belief to the contrary, the ability to deal efficiently with such worst case scenarios may not be a prerequisite for solving quickly. It will compare on the one hand AC-2001, which has an optimal worst case time-complexity and is considered efficient, and on the other AC-3_d, which is not optimal when it comes to its worst case time-complexity, but which has a better space-complexity than AC-2001. Both algorithms will be compared for MAC search and for *stand alone arc-consistency* (the task of making a single CSP arc-consistent). For stand alone arc-consistency AC-3_d is the better algorithm when it comes to time but there is no clear winner when it comes to minimising the number of checks. For search the results are more interesting. MAC-2001 is by far the better algorithm when it comes to minimising the number of checks. However, MAC-3_d is considerably faster on average. For difficult random problems, that took between minutes and 1.5 hour to solve, MAC-3_d was about 1.5 times faster on average than MAC-2001. As soon as MAC-2001 starts to become successful in avoiding the duplication of many checks it begins to invest much more additional solution time. These observations suggest that being worst case optimal may come at a price of being less efficient on average in search and that algorithms like MAC-3_d are promising.

Contents

List of Figures

Chapter 1

Introduction

Arc-consistency algorithms significantly reduce the size of the search space of Constraint Satisfaction Problems (CSPs) at low costs. They are the work horse of many backtrack searchers that Maintain Arc-Consistency during search (MAC) [Sabin and Freuder, 1994].

Currently, there seems to be a shared belief in the constraint satisfaction community that, to be efficient, arc-consistency algorithms should have an optimal worst case time-complexity [Bessière *et al.*, 1995; Bessière and Régin, 2001; Zhang and Yap, 2001]. Arc-consistency and MAC algorithms that are optimal in their worst case time-complexity require a space-complexity of at least $\mathbf{O}(ed)$ to create data structures to remember their support-checks. MAC algorithms need a $\mathbf{O}(ed \min(n, d))$ space-complexity to *maintain* these data structures. As usual, n is the number of variables in the CSP, d is the maximum domain size of the variables and e is the number of constraints. The bookkeeping that is involved with these data structures is a significant overhead. There is also experimental evidence that arc-consistency and MAC algorithms with a low $\mathbf{O}(e + nd)$ space-complexity can be good even if they cannot remember all their support-checks and—as a consequence—have to repeat them [van Dongen, 2002b; 2002c; 2002d].

In this report we shall provide evidence to support the claim that good arc-consistency algorithms do not need to have an optimal worst case time-complexity. We shall experimentally compare two AC-3 based arc-consistency algorithms [Mackworth, 1977]. The first algorithm is Bessière and Régin's AC-2001 [Bessière and Régin, 2001]. AC-2001 has an optimal $\mathbf{O}(ed^2)$ worst case time-complexity, has a $\mathbf{O}(ed)$ space-complexity, and is considered good on average [Bessière and Régin, 2001]. The second algorithm is AC-3_d [van Dongen, 2002b; 2002c; 2002d]. AC-3_d has a worse $\mathbf{O}(ed^3)$ worst case time-complexity than AC-2001 but it has a better $\mathbf{O}(e + nd)$ space-complexity. Results from a preliminary comparison with AC-7, another optimal arc-consistency algorithm [Bessière *et al.*, 1995], indicate that AC-3_d is promising [van Dongen, 2002c; 2002d]. We shall compare both algorithms for MAC search and for *stand alone arc-consistency*. Here stand alone arc-consistency is the task of making a single CSP arc-consistent or decide that this is not possible. As part of our presentation we shall introduce some notation to conveniently define ordering heuristics.

Our results for search demonstrate that for as far as support-checks are concerned MAC-2001 was by far the better algorithm. More importantly, however, MAC-3_d was significantly better on

average for wall time. MAC-3_d was almost between 1.25 and 5.75 times faster on average than MAC-2001, whereas MAC-2001 was never significantly faster than MAC-3_d. For problems that took a solution time of the order of magnitude of an hour, MAC-3_d was about 1.5 times faster on average than MAC-2001. Our results indicate that if checks are cheap—and they almost always are—then one should prefer an algorithm like MAC-3_d. For stand alone arc-consistency the results were less clear. AC-3_d was the better algorithm when it came to wall time. For minimising the number of consistency-checks there was no clear winner.

Finally, we shall present proof that MAC-2001 has a $\mathbf{O}\left(ed\min(n,d)\right)$ space-complexity. This result does not seem to have been noticed before.

The results presented in this report are important because of the following. Since the introduction of Mohr and Henderson's AC-4 [Mohr and Henderson, 1986], most work in arc-consistency research has been focusing on the design of better algorithms that do not re-discover (algorithms that do not repeat checks). Our key insight is that it is only possible to avoid re-discoveries at the price of a large additional bookkeeping. To forsake the bookkeeping at the expense of having to re-discover may improve search. This insight may lead to the design of new classes of arc-consistency and MAC algorithms that are not only competitive but may also, like AC-$3_d$and MAC-3_d, have the advantage of a better space-complexity because they do not have to remember all their checks. AC-3_d and MAC-3_d are the first known efficient algorithms from these classes.

Finally, it should be noted that since AC-3_d can be considered as a specialisation of AC-3, our results imply that AC-3 with proper heuristics is also efficient. This observation goes in against all current belief in constraint satisfaction.

Chapter 2

Constraint Satisfaction

A binary *constraint* C_{xy} between variables x and y is a subset of the cartesian product of the domains $D(x)$ of x and $D(y)$ of y. A value $v \in D(x)$ is *supported* by $y \in D(y)$ if $(v, w) \in C_{xy}$. Similarly, $w \in D(y)$ is supported by $v \in D(x)$ if $(v, w) \in C_{xy}$. If $v \in D(x)$ is supported by $w \in D(y)$ then we shall also say that $v \in D(x)$ is supported by y.

A *Constraint Satisfaction Problem* (CSP) is a tuple $\mathcal{C} = (X, D, C)$, where X is a set of variables, $D(\cdot)$ is a function mapping each $x \in X$ to its non-empty domain, and C is a set of constraints between variables in subsets of X. We shall only consider CSPs whose constraints are binary. $\mathcal{C}$ is called *arc-consistent* if its domains are non-empty and for each $C_{xy} \in C$ it is true that every $v \in D(x)$ is supported by y and that every $w \in D(y)$ is supported by x. A *support-check* (consistency-check) is a test to find out if two values support each other.

The *tightness* of the constraint C_{xy} between x and y is defined as $1 - | C_{xy} | / | D(x) \times D(y) |$, where $\cdot \times \cdot$ denotes cartesian product. The *density* of a CSP is defined as $2e/(n^2 - n)$, for $n > 1$.

A MAC solver is a backtracker that maintains arc-consistency during search [Sabin and Freuder, 1994]. MAC-i is a MAC solver that uses arc-consistency algorithm AC-i to maintain its arc-consistency.

Chapter 3

Operators for Selection Heuristics

3.1 Introduction

In this chapter we shall introduce some notation to unambiguously describe and "compose" relations. Such notation is essential to describe selection heuristics for variables and arcs in MAC searchers but may also be useful in other domains. The notation will significantly simplify our description of selection heuristics.

We shall first provide a foundation for combining and constructing new orders from other relations and then use these foundations to define existing and variable selection and arc selection heuristics.

3.2 Composition of Selection Heuristics

In this section we shall recall the basic definitions of the notion of a *linear quasi-order* and that of a *(linear) order* and define a new notation for constructing a new order from an existing linear quasi-order and an existing order. The idea of combining orders is strongly influenced by idea by Collart, Kalkbrener, and Mall to combine partial orders on terms [Collart *et al.*, 1997].

Let T be a set. A relation on T is called a *quasi-order* if it is reflexive and transitive. Quasi-orders $\preceq$ may allow for situations where $v \preceq w \wedge w \preceq v \wedge v \neq w$. A good example of a quasi-order is the divisability relation $\cdot|\cdot$ on $\mathbb{N}$. For example, we have $2|6 \wedge 3|6 \wedge \neg 2|3 \wedge \neg 3|2$. Another good example of a quasi-order is the relation $\preceq$ on $\mathbb{Z}^2$ which is defined as $(v, w) \preceq (v', w')$ if and only if $v + w \leq v' + w'$. A relation, $\prec$, on T is called *linear* if $v \prec w \vee w \prec v$ for all $v, w \in T$. A quasi-order $\preceq$ is called a *partial order* if $v \preceq w \wedge w \preceq v$ implies $v = w$ for all v and $w \in T$. An *order* (also called a *linear order*) is a partial order that is also a linear quasi-order. A good example of orders are the relations $\leq$ and $\geq$ on $\mathbb{N}$.

For many selection heuristics it is desirable that they always select a single unique optimum from the a given set of objects. Such heuristics are equivalent to orders. Similarly, useful selection heuristics that allow for ties are equivalent to linear quasi-orders. We shall now define an operator to *compose* a linear quasi-order, $\preceq_1$, and an order, $\preceq_2$, into a new order that may be viewed as the order that uses $\preceq_1$ and breaks ties using $\preceq_2$.

Let $\preceq_1$ be a linear quasi-order and let $\preceq_2$ be a order on T. By $\preceq_2 \bullet \preceq_1$ we will mean the *composition* of $\preceq_2$ and $\preceq_1$. This composition may be viewed as a *refinement* of $\preceq_1$ by $\preceq_2$. It is the unique order on T that is defined as follows:

$$v \preceq_2 \bullet \preceq_1 w \iff (v \preceq_1 w \wedge \neg w \preceq_1 v) \vee (v \preceq_1 w \wedge w \preceq_1 v \wedge v \preceq_2 w).$$

In words, if v is smaller than w with respect to $\preceq_1$ or vice versa then $\preceq_1$ will determine the result of the composition. Otherwise, i.e. if v and w are equal with respect to $\preceq_1$ then $\preceq_2$ will be used to determine the ordering of the composition.

Note that if $\preceq_1$ is an order then $\preceq_2 \bullet \preceq_1$ is equal to $\preceq_1$. If $\preceq_1$ is not an order then $\preceq_2 \bullet \preceq_1$ may be viewed as the order that first uses $\preceq_1$ and "breaks ties" using $\preceq_2$. Composition associates to the left, i.e. $\preceq_3 \bullet \preceq_2 \bullet \preceq_1$ is equal to $(\preceq_3 \bullet \preceq_2)\bullet \preceq_1$. The symbol "$\bullet$" was chosen for order composition because it is reminiscent of "$\circ$" for function composition.

Let $\preceq$ be a linear quasi-order on Y, let v be a variable, and let $f :: T \mapsto Y$ be a function. Then $\otimes^f_{\preceq}$ is the unique order on T which is defined as follows:

$$v \otimes^f_{\preceq} w \iff f(v) \preceq f(w).$$

3.3 Operators for Variable and Arc Selection

We are now in a position to define some well known variable selection heuristics and arc selection heuristics very compactly. We shall first define variable selection heuristics and then define arc selection heuristics.

To define the variable ordering heuristics, let $\delta_o(v)$ be the original degree of v, let $\delta_c(v)$ be the current degree of v, let $\kappa(v)$ be given by $|D(v)|$, and let $\#(v)$ be the unique number of v according to some preference. We shall use these functions and the standard orders $\leq$ and $\geq$ on $\mathbb{N}$ to define orders on the variables. In the remainder of this report we shall assume that $\#(v) < \#(w)$ is true if and only if v is lexicographically smaller than w.

The well known minimum domain size heuristic with a lexicographical tie breaker is given by $\otimes^{\#}_{\leq} \bullet \otimes^{\kappa}_{\leq}$. The Brelaz heuristic [Gent *et al.*, 1996] with a lexicographical tie breaker is given by $\otimes^{\#}_{\leq} \bullet \otimes^{\delta_c}_{\geq} \bullet \otimes^{\kappa}_{\leq}$. An ordering on the maximum original degree with a lexicographical tie breaker is given by $\otimes^{\#}_{\leq} \bullet \otimes^{\delta_o}_{\geq}$. Note that we only need one of the orders $\leq$ and $\geq$ because $a \leq b \iff -a \geq -b$. With this equivalence the Brelaz heuristic with a lexicographical tie breaker can also be defined as $\otimes^{\#}_{\leq} \bullet \otimes^{-\delta_c}_{\leq} \bullet \otimes^{\kappa}_{\leq}$.

As an exercise, the reader is invited to define some other useful variable ordering heuristics.

For arc selection heuristics we need a few more ingredients. Two useful operators are the projection operators π_1 and π_2 which are defined as $\pi_1((\, v, w\,)) = v$ and $\pi_2((\, v, w\,)) = w$. The following defines a lexicographical ordering heuristic:

$$\otimes^{\#\circ\pi_2}_{\leq} \bullet \otimes^{\#\circ\pi_1}_{\leq}.$$

Here, $\cdot \circ \cdot$ denotes function compositions, so that $\# \circ \pi_i((\, v, w\,)) = \#(\pi_i((\, v, w\,)))$. As a final example, consider the following order:

$$\otimes^{\#\circ\pi_2}_{\leq} \bullet \otimes^{\delta_c\circ\pi_2}_{\geq} \bullet \otimes^{\kappa\circ\pi_2}_{\leq} \bullet \otimes^{\#\circ\pi_1}_{\leq} \bullet \otimes^{\delta_c\circ\pi_1}_{\geq} \bullet \otimes^{\kappa\circ\pi_1}_{\leq}. \tag{3.1}$$

The order defined in Equation (3.1) turns out to be an excellent dynamic arc selection heuristic for AC-3_d.[1] It is the same order as the order $\preceq'$ which some people may define as follows:

$$(v, w) \preceq' (v', w') \iff \begin{cases} true & \text{if } \kappa(v) < \kappa(v'); \\ false & \text{else if } \kappa(v') > \kappa(v); \\ true & \text{else if } \delta_c(v) > \delta_c(v'); \\ false & \text{else if } \delta_c(v') < \delta_c(v); \\ true & \text{else if } \#(v) < \#(v'); \\ false & \text{else if } \#(v') > \#(v); \\ true & \text{else if } \kappa(w) < \kappa(w'); \\ false & \text{else if } \kappa(w') > \kappa(w); \\ true & \text{else if } \delta_c(w) > \delta_c(w'); \\ false & \text{else if } \delta_c(w') < \delta_c(w); \\ true & \text{else if } \#(w) \leq \#(w'); \\ false & \text{otherwise.} \end{cases}$$

More difficult than writing a definition for this order is to describe it in words. It is hoped that the reader agrees that the notation in Equation (3.1) is not only more compact but also is easier to comprehend.

[1] A better heuristic still remains to be found.

Chapter 4

Related Literature

In 1977, Mackworth presented an arc-consistency algorithm called AC-3 [Mackworth, 1977]. AC-3 has a $\mathbf{O}\,(ed^3)$ bound for its worst case time-complexity [Mackworth and Freuder, 1985]. AC-3 has a $\mathbf{O}\,(e + nd)$ space-complexity. AC-3 cannot remember all its support-checks. It uses *arc-heuristics* to repeatedly select and remove a tuple, $(\,x, y\,)$, from a data structure called a *queue* and to use the constraint between x and y to *revise* the domain of x. Here, to revise the domain of x using the constraint between x and y means to remove the values from $D(x)$ that are not supported by y. These arc-heuristics determine the constraint that will be used for the next support-check. Besides these arc-heuristics there are also *domain-heuristics*. These heuristics, if given the constraint that will be used for the next support-check, determine the values that will be used for the next support-check. Depending on the outcome of the revision process new arcs may be added to the queue. The interested reader is referred to [Mackworth, 1977] for a detailed description of AC-3.

Wallace and Freuder pointed out that arc-heuristics can influence the efficiency of arc-consistency algorithms [Wallace and Freuder, 1992]. Similar observations were made by Gent *et al.* [Gent *et al.*, 1997].

Bessière and Régin presented AC-2001, which is based on AC-3 [Bessière and Régin, 2001] (see also [Zhang and Yap, 2001] for a similar algorithm). AC-2001 revises one domain at a time. The main difference between AC-3 and AC-2001 is that AC-2001 uses a lexicographical domain-heuristic and that for each variable x, for each $v \in D(x)$ and each constraint between x and another variable y it remembers the last support for $v \in D(x)$ with y so as to avoid repeating checks that were used before to find support for $v \in D(x)$ with y. AC-2001 has an optimal upper bound of $\mathbf{O}\,(ed^2)$ for its worst case time-complexity and its space-complexity is $\mathbf{O}\,(ed)$. Bessière and Régin found that AC-2001 behaves well on average. Together with Freuder they also observed that AC-3 is a good alternative for stand alone arc-consistency if checks are cheap and CSPs are under-constrained [Bessière *et al.*, 1999]. However, they also observed that AC-3 was significantly slower than AC-7 for over-constrained CSPs and CSPs in the phase transition. Similar observations were made in [Bessière and Régin, 2001] where it was observed that AC-3 was good for under-constrained CSPs but slower than AC-6 and AC-2001 for over-constrained CSPs and CSPs in the phase-transition. AC-3's incapability to make inference allows it to do well for easy CSPs.

It seems to have gone unnoticed so far that MAC-2001 has a $\mathbf{O}(ed\min(n,d))$ space-complexity. The reason for this space-complexity is that MAC-2001 has to *maintain* AC-2001's $\mathbf{O}(ed)$ data structures during search. These data structures consist of a counter for each constraint-value pair to remember the last support for that value [Bessière and Régin, 2001]. To maintain these data structures, MAC-2001 has to save the counters of the current and future variables after each assignment to the current variable and to restore them upon backtracking. The only ways to save and restore the counters seem to be one of the following three methods:

1. Save all relevant counters once before arc-consistency. Upon backtracking these counters have to be restored. This requires a $\mathbf{O}(ned)$ space-complexity because $\mathbf{O}(ed)$ data structures may have to be saved n times.

2. Save each counter before it is incremented and count the number of increments, i, that were carried out during the arc-consistency call immediately after the assignment to the current variable. Upon backtracking, the increments can be undone by restoring i counters in the reverse order. This comes at the price of a space-complexity of $\mathbf{O}(ed^2)$ because each of the $2ed$ counters may have to be saved d times.

3. A combination of the previous two.

Combining these results we have a $\mathbf{O}(ed\min(n,d))$ space-complexity. Christian Bessière (private communication) agreed that this analysis is, indeed, correct and that he had implemented MAC-2001 using Method 2.

Rick Wallace experimentally found that AC-3 was always always better than Mohr and Henderson's AC-4, which has an optimal worst case time-complexity but which is almost always slow on average due to the maintenance of it huge data structures [Mohr and Henderson, 1986; Wallace, 1993]. These findings suggest that it is not always an absolute necessity for an arc-consistency algorithm to have an optimal time-complexity.

Similar observations were made in an experimental comparison between AC-3, AC-7, and AC-3_d, which is a cross-breed between Mackworth's AC-3 and Gaschnig's DEE [Mackworth, 1977; Gaschnig, 1978; Bessière *et al.*, 1995; van Dongen, 2002a; 2002d; 2002c]. AC-7 is optimal. The only difference between AC-3 and AC-3_d is that AC-3_d sometimes takes two arcs out of the queue and simultaneously revises *two* domains with a *double-support* domain-heuristic. AC-3_d only simultaneously revises two domains if its arc-heuristic selects the arc (x,y) from the queue and if, at the time of that selection, the reverse arc (y,x) also turns out to be in the queue. AC-3_d's double-support heuristic prefers checks between two values each of which are not yet known to be supportable. The interested reader is referred to [van Dongen, 2002d] for a detailed description of AC-3_d's implementation. It should be pointed out that if AC-3_d's double-support heuristic is replaced by a call to Mackworth's revise to revise one domain and one more call to revise the other domain if the first call did not result in an inconsistency, then the resulting algorithm, requires more time on average than AC-3_d. A big advantage of MAC-3_d is that it has a low $\mathbf{O}(e+nd)$ space-complexity, strictly smaller than AC-2001's $\mathbf{O}(ed)$, and that MAC-3_d does not require additional data structures during search. Note that if $n \approx d$ then MAC-3_d's space-complexity is the "square root" of that of MAC-2001. To see why this is true,

first notice that there can be $n \times (n-1)/2$ constraints and that as a consequence $e \in \mathbf{O}(n^2)$. Next notice that if $n \approx d$ then $\mathbf{O}(e + nd)$ becomes $\mathbf{O}(n^2)$ and $\mathbf{O}(ed \min(n, d))$ becomes $\mathbf{O}(n^4)$. For two-variable CSPs the double-support heuristic is optimal. It is about twice as efficient as the lexicographical heuristic if the domain sizes of the variables are about equal [van Dongen, 2002a; 2002b]. In our comparison we found that for under-constrained CSPs AC-3_d was slightly worse than AC-3 in wall time. AC-3 was significantly slower than AC-3_d for over-constrained CSPs and CSPs in the phase transition. This is consistent with Bessière, Freuder and Régin's findings [Bessière *et al.*, 1999]. In checks AC-3_d was significantly worse than AC-7 in the phase transition region but in wall time both algorithms performed about equally well. For all other problems AC-3_d turned out to be better both in wall time and checks. These are surprising results because AC-3_d repeats support-checks, whereas AC-7 does not. They support the thesis of this report that, despite common belief to the contrary, arc-consistency algorithms can be efficient if they are not worst case optimal. In the following chapter we shall see more evidence to support this claim.

AC-3_d's best arc-heuristic found so far is the heuristic described in Equation (3.1). This heuristic is relatively expensive. For example, between 15% and 25% of MAC-3_d's time is spent on arc selection and this does not include the time to see if the reverse arc is also in the queue and the time to modify the queue. However, the heuristic is good because it leads to relatively many selections where two domains can be revised simultaneously. Cheaper heuristics like the lexicographical heuristic usually do not lead to improvements for MAC-3_d. For example, a lexicographical arc-heuristic almost forces MAC-3_d to degenerate to MAC-3 because most of the arcs that are selected with this heuristic cannot be used for simultaneous revision. What is worse is that if AC-3_d uses the expensive heuristic and requires r_d double revisions and r_s single revisions then AC-3_d with the cheap heuristic, almost always requires $r'_d < r_d$ double revisions and more than $r_s + 2(r_d - r'_d)$ single revisions, i.e. the $r_d - r'_d$ fewer double revisions that are carried out using the cheap heuristic are traded in for more than $2(r_d - r'_d)$ single revisions. This bad news because a single double revision of two domains is faster on average than two single revisions of those domains.

Chapter 5

Experimental Results

5.1 Introduction

In this chapter we shall experimentally compare AC-2001 and AC-3_d for MAC search and for stand alone arc-consistency. To compare the algorithms we used Christian Bessière's implementation of MAC-2001 and our own implementation of MAC-3_d. Both solvers were members of the real-full-look-ahead family. Bessière's implementation was a specialised version for random problems. This is why we only considered random problems for our comparison. However, the results from this chapter are in line with and strengthen the observations that MAC-3_d is good [van Dongen, 2002d; 2002c]. Our own implementation was an improvement of the implementation used for the comparison described in [van Dongen, 2002c; 2002d]. For some problems these improvements resulted in a speed up of 30% to 40%. It should be noticed that, as a consequence of the improved implementation, the results described in [van Dongen, 2002c; 2002d] are outdated in the sense that they describe MAC-3_d as being slower than it is at the time of writing this report.

To ensure that both searchers visited the same nodes in the search tree they were equipped with the same dom/deg variable ordering heuristic. Using the notation introduced in Chapter 3 this heuristic is given by $\otimes^{\#}_{\leq} \bullet \otimes^{f}_{\leq}$, where $f(v) = \kappa(v)/\delta_o(v)$. Bessière's implementation of MAC-2001 came with only one arc-heuristic, the lexicographical heuristic. MAC-3_d was equipped with the arc-heuristic defined as follows:

$$\otimes^{\#\circ\pi_2}_{\leq} \bullet \otimes^{\delta_c\circ\pi_2}_{\geq} \bullet \otimes^{\kappa\circ\pi_2}_{\leq} \bullet \otimes^{\#\circ\pi_1}_{\leq} \bullet \otimes^{\delta_c\circ\pi_1}_{\geq} \bullet \otimes^{\kappa\circ\pi_1}_{\leq}.$$

The problems were generated as follows. For each combination (C, T) of average density C and uniform tightness T in $\{ (i/20, j/20) : 1 \leq i, j \leq 19 \}$ 50 random CSPs were generated with $n = 30$ variables and uniform domain size $d = 30$. Next we computed the average number of checks and the average time required for C and T by MAC-2001 and MAC-3_d for the tasks of making the problem arc-consistent before starting search and for deciding the satisfiability of each problem using MAC search. All problems were run to completion. Frost *et al.*'s model B [Gent *et al.*, 2001] random problem generator was used to generate the problems (`http://www.lirmm.fr/~bessiere/generator.html`).

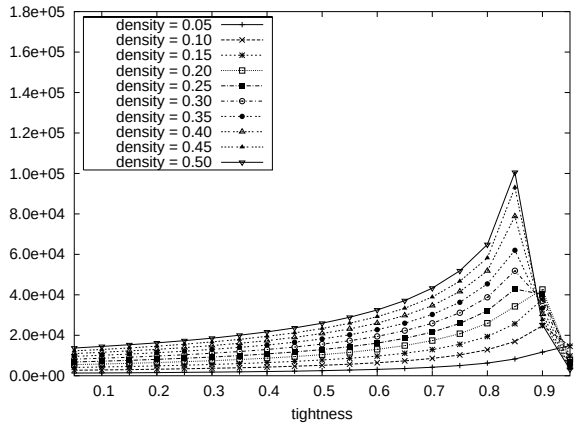

Figure 5.1: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001.

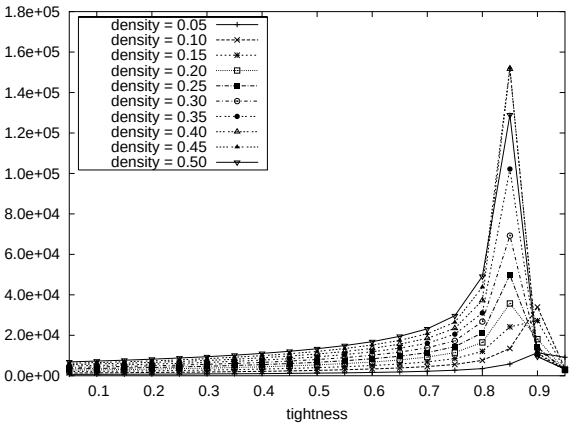

Figure 5.2: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-3$_d$.

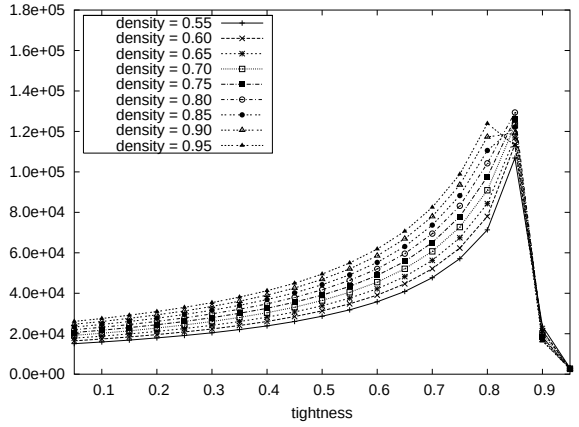

Figure 5.3: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $0.5 < C$, AC-2001.

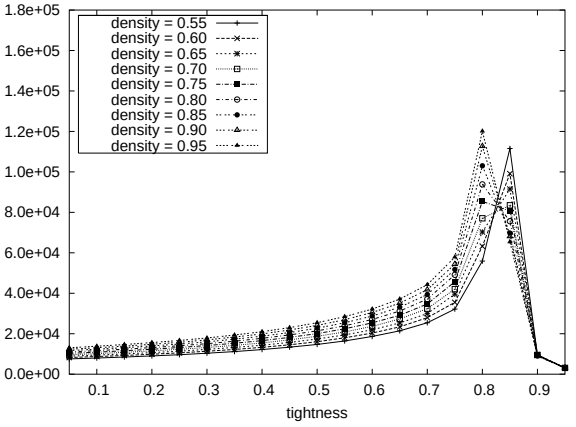

Figure 5.4: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $0.5 < C$, AC-3$_d$.

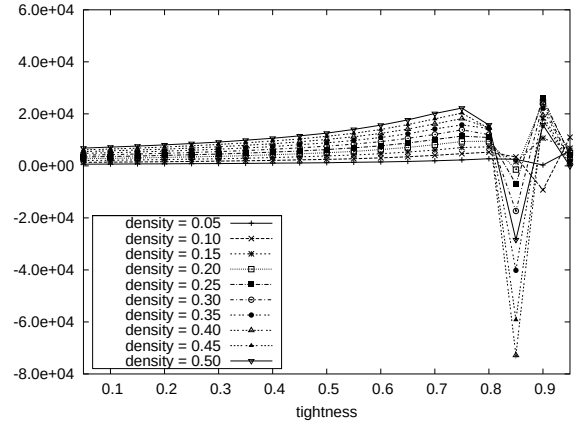

Figure 5.5: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001 – AC-3$_d$.

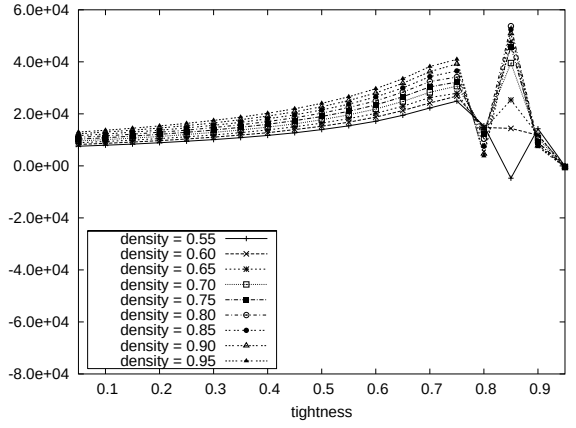

Figure 5.6: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $0.5 < C$, AC-2001 – AC-3$_d$.

The test was carried out in parallel on 50 identical machines. Each machine was associated with a unique identifier in the range from 1 through 50. For each machine random problems were generated for each combination of density and tightness. The CSP generator on a particular machine was started with the seed given by the identifier of that machine. All problems fitted into main memory and no swapping occurred.

In this chapter we shall only present the findings for $n = d = 30$. We also carried out the comparison for $n = d = 10$ and $n = d = 20$ (See Appendix A.1 for the results for $n = d = 20$). The results for these comparisons were in line with the case $n = d = 30$. The only difference was, of course, that less time and fewer checks were required.

5.2 Stand alone Arc-Consistency

The results for stand alone arc-consistency and checks are depicted in Figures 5.1–5.6. Figures 5.5 and 5.6 depict the difference between the average number of checks. On average AC-3_d requires fewer checks almost everywhere except if tightness is between 80% and 90% and if density is below 50%. For tightness between 5% and 70% the ratio between the average number of checks required by AC-2001 and by AC-3_d is about 2 (Figures A.35 and A.36 depict the ratio graphically).

The results for checks and stand alone arc-consistency are remarkable because AC-2001 does not repeat support-checks whereas AC-3_d does. It should be interesting to theoretically investigate the reason for this counter-intuitive phenomenon. An initial theoretical investigation for 2-variable CSPs has provided interesting results [van Dongen, 2002a; 2002b]. For the moment the general case seems to be too difficult.

The results for time and stand alone arc-consistency are depicted in Figures 5.7–5.12. These figures depict the average time that was required by AC-2001 and AC-3_d. Only for very easy problems is AC-2001 as good as AC-3_d. For the remaining problems AC-3_d is between 1.5 and 2 times faster.

The results for stand alone arc-consistency are clearly in favour of AC-3_d. Only for a small fraction of the combinations of density and tightness did AC-2001 do better on average in checks than AC-3_d. For the remaining vast majority of combinations of density and tightness AC-3_d was better on average in checks. On average and modulo the accuracy of the timer, AC-2001 was never faster than AC-3_d but it was significantly slower for the majority of the problems.

There is no such thing as "the" best arc-consistency algorithm. However, the results presented in this section indicate that for random problems and stand alone arc-consistency AC-3_d is a good choice and a good substitute for AC-2001.

5.3 Maintain Arc-Consistency

Figures 5.13 –5.18 depict the average solution time of MAC-2001 and MAC-3_d. These pictures seem to suggest that MAC-3_d is faster on average in solving random CSPs than MAC-2001.

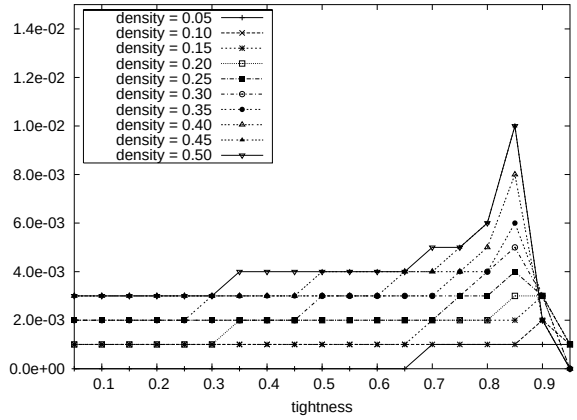

Figure 5.7: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-2001.

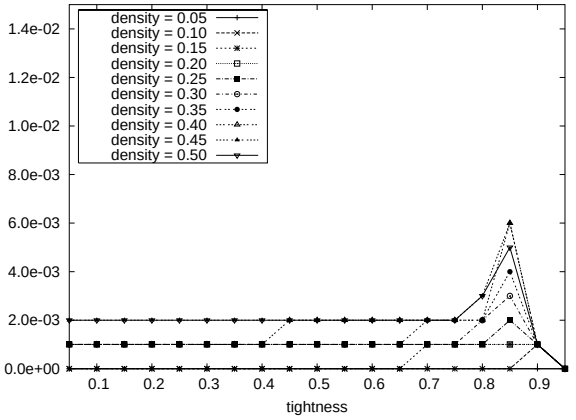

Figure 5.8: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-3$_d$.

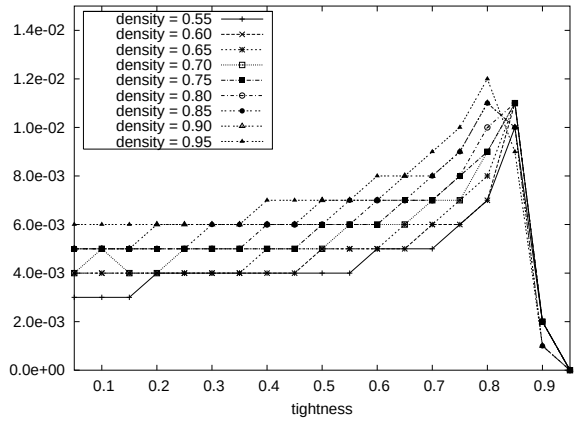

Figure 5.9: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $0.5 < C$, AC-2001.

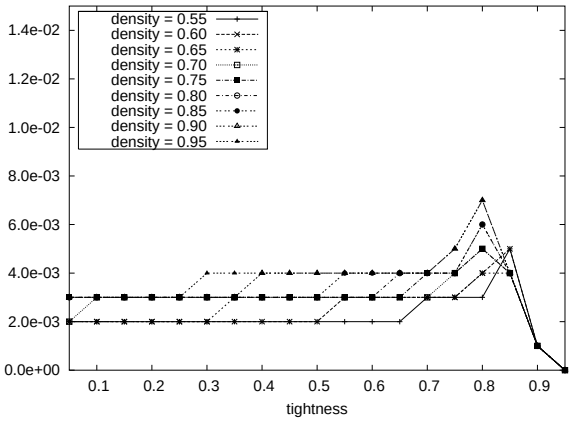

Figure 5.10: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $0.5 < C$, AC-3$_d$.

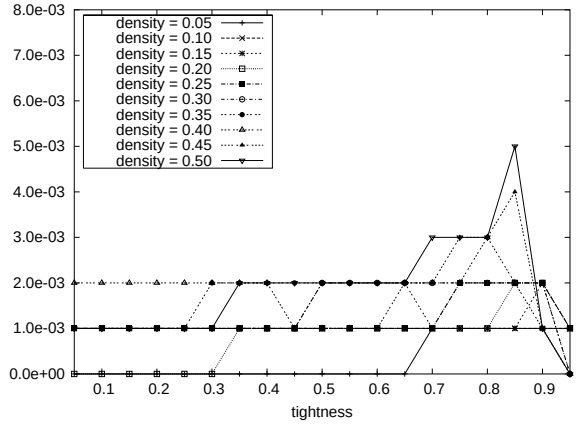

Figure 5.11: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-2001 – AC-3$_d$.

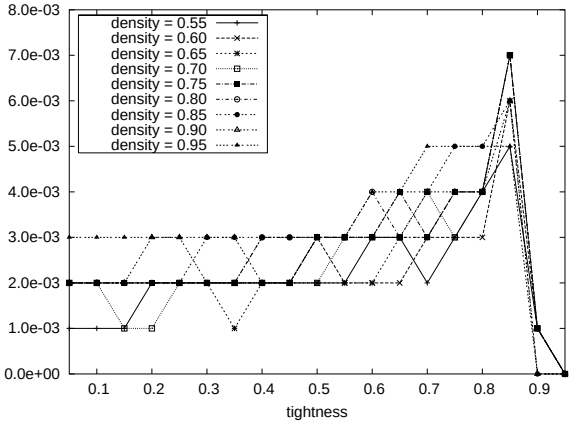

Figure 5.12: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $0.5 < C$, AC-2001 – AC-3$_d$.

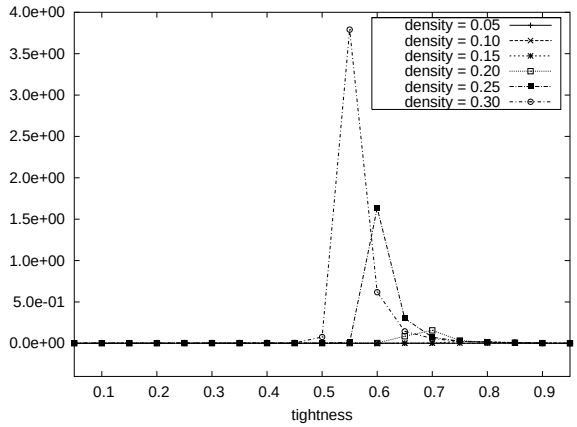

Figure 5.13: $n = 30$, $d = 30$, Search: Time, $C \leq 0.3$, MAC-2001.

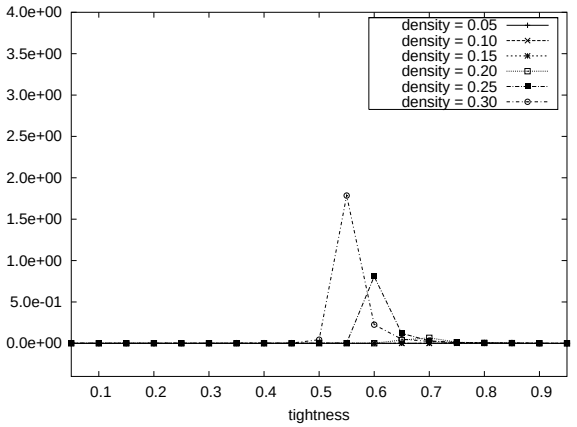

Figure 5.14: $n = 30$, $d = 30$, Search: Time, $C \leq 0.3$, MAC-3$_d$.

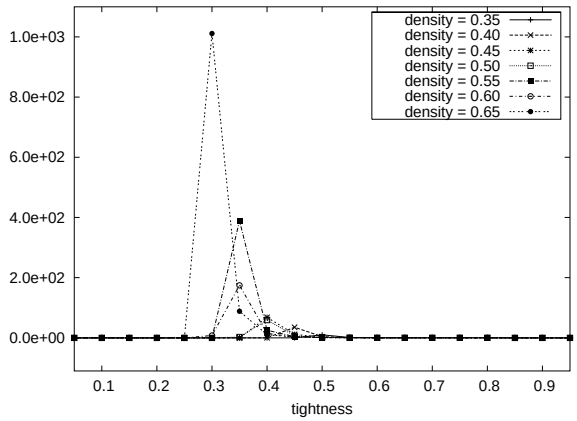

Figure 5.15: $n = 30$, $d = 30$, Search: Time, $0.3 < C < 0.7$, MAC-2001.

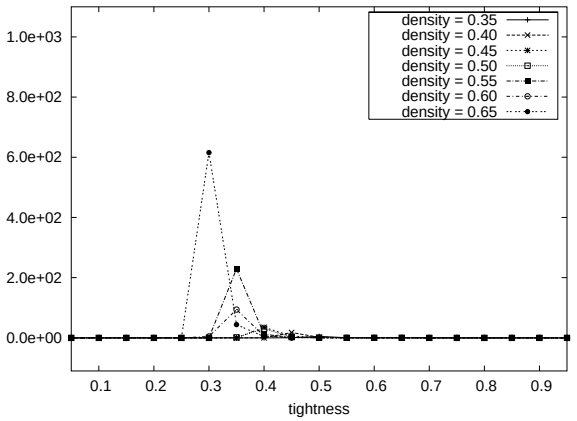

Figure 5.16: $n = 30$, $d = 30$, Search: Time, $0.3 < C < 0.7$, MAC-3$_d$.

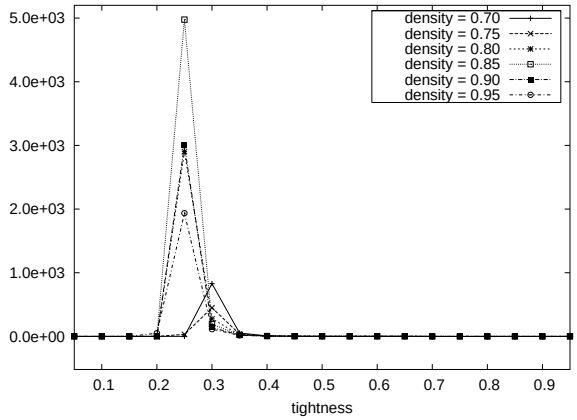

Figure 5.17: $n = 30$, $d = 30$, Search: Time, $0.7 \leq C$, MAC-2001.

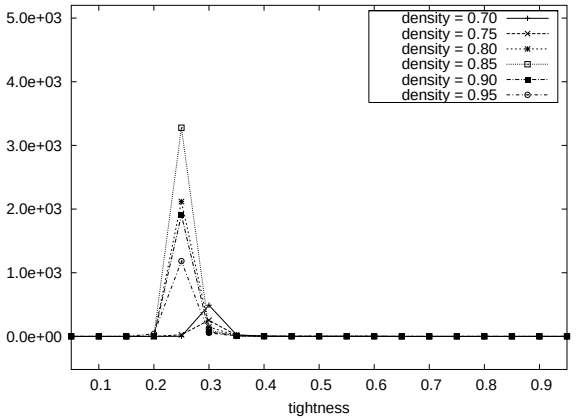

Figure 5.18: $n = 30$, $d = 30$, Search: Time, $0.7 \leq C$, MAC-3$_d$.

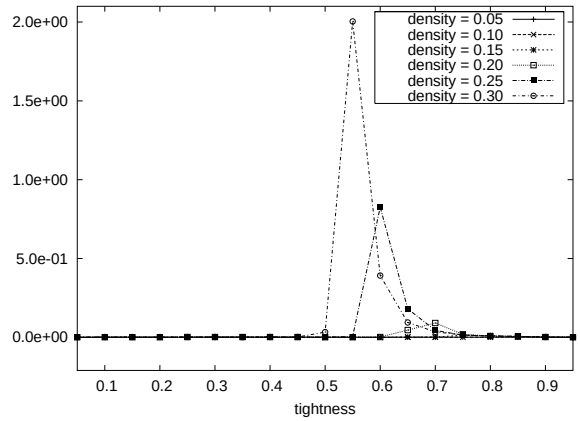

Figure 5.19: $n = 30$, $d = 30$, Search: Time, $C \leq 0.3$, MAC-2001 – MAC-3$_d$.

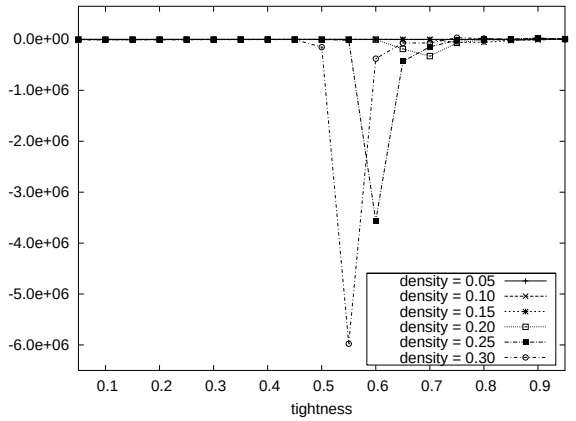

Figure 5.20: $n = 30$, $d = 30$, Search: Checks, $C \leq 0.3$, MAC-2001 – MAC-3$_d$.

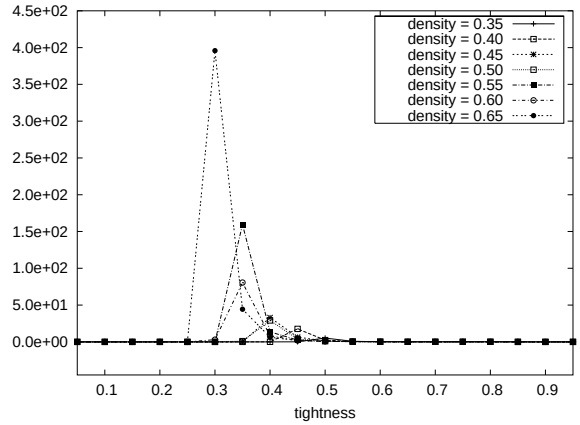

Figure 5.21: $n = 30$, $d = 30$, Search: Time, $0.3 < C < 0.7$, MAC-2001 – MAC-3$_d$.

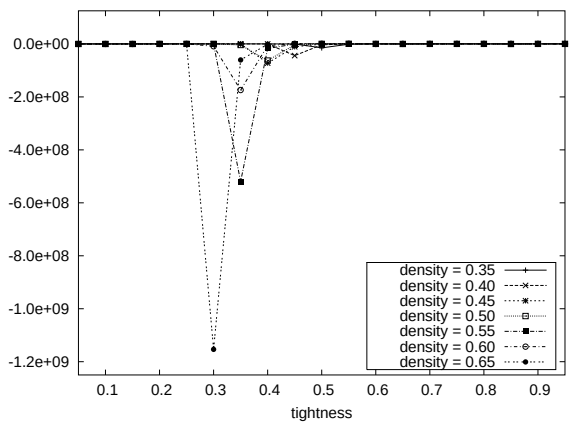

Figure 5.22: $n = 30$, $d = 30$, Search: Checks, $0.3 < C < 0.7$, MAC-2001 – MAC-3$_d$.

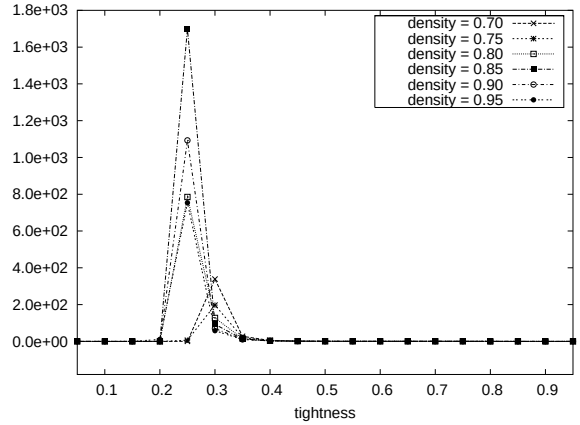

Figure 5.23: $n = 30$, $d = 30$, Search: Time, $0.7 \leq C$, MAC-2001 – MAC-3$_d$.

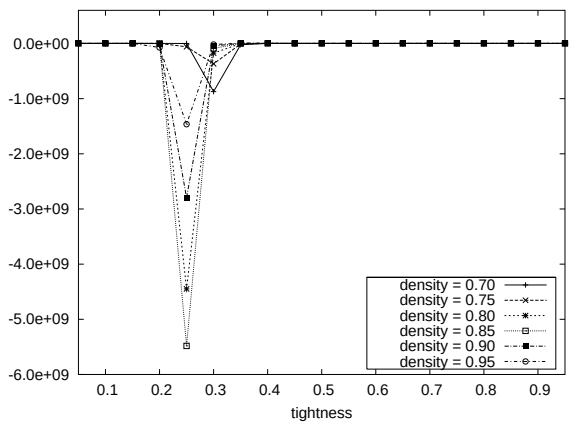

Figure 5.24: $n = 30$, $d = 30$, Search: Checks, $0.7 \leq C$, MAC-2001 – MAC-3$_d$.

This is confirmed by Figures 5.19, 5.21 and 5.23 which depict the difference between the average solution time required by MAC-2001 and MAC-3_d. For certain problems, MAC-2001 requires more than thousand seconds more than MAC-3_d. However, these graphs do not reveal everything. Closer inspection of the data demonstrated that there were only two combinations of density and tightness for which MAC-2001 was better on average. For $(C, T) = (0.95, 0.05)$ MAC-2001 required 0.016 seconds as opposed to MAC-3_d's 0.017 seconds. For $(C, T) = (0.85, 0.20)$ it required 0.105 seconds, whereas MAC-3_d required 0.109 seconds. These differences are negligible. For the remaining cases, MAC-3_d was always at least as good as MAC-2001 but was usually better on average. The ratio between the average solution times require by MAC-2001 and MAC-3_d is depicted in Figures 5.25 and 5.26. For those solution times where MAC-3_d required 0 seconds (within the accuracy of the timer) a ratio of 1 was assumed. For most problems MAC-3_d turned out to be between 1.25 and 5.75 times faster.

To find the reason why MAC-2001 was slower than MAC-3_d we also have to study Figures 5.20, 5.22 and 5.24. These figures depict the difference between the number of checks that were required by MAC-2001 and MAC-3_d. When compared to MAC-3_d, MAC-2001 starts to lose out in time as sooon as it starts becoming successful in saving many checks. The time that is traded in for savings in checks is roughly proportional to the number of checks as soon as there are many. For millions of checks that MAC-2001 saves more than MAC-3_d it loses seconds in solution time. Our experimental results suggest the hypothesis that when checks are cheap MAC-3_d's approach to duplicate work seems better when it comes to solution time.

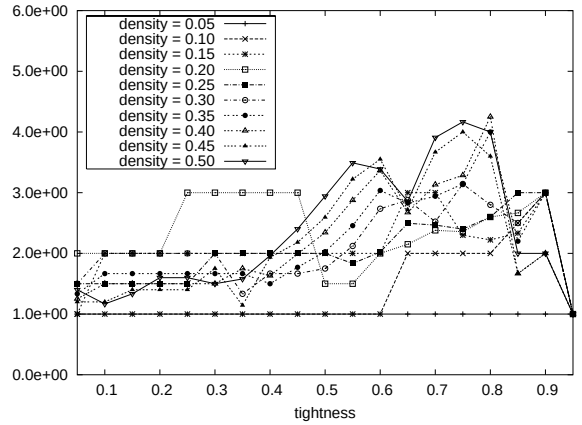

Figure 5.25: $n = 30$, $d = 30$, Search: Time, $C \leq 0.5$, MAC-2001/MAC-3_d.

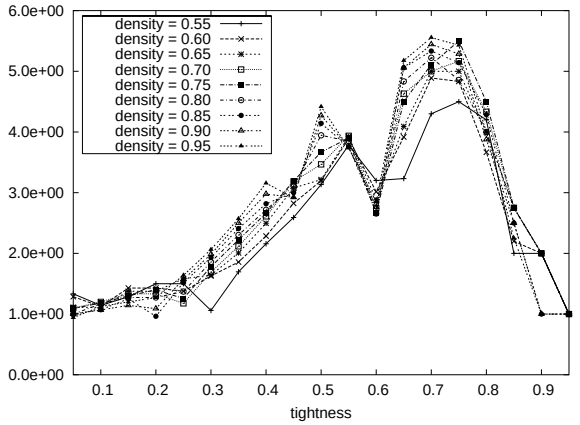

Figure 5.26: $n = 30$, $d = 30$, Search: Time, $0.5 < C$, MAC-2001/MAC-3_d.

5.4 Statistical Analysis

In this section we shall analyse the data for MAC search in more detail. We will verify our hypothesis that MAC-2001 loses time as soon as it starts saving checks. To do this we defined two stochastic variables: A, the *difference in time* between MAC-2001 and MAC-3_d, and B, the

difference in checks between MAC-2001 and MAC-3_d. Our experimental data gave us $N = 50 \times 19 \times 19 = 18050$ samples (A_i, B_i) of (A, B), $1 \leq i \leq N$. To determine the correlation between A and B using these samples we used the *Pearson Product-Moment Correlation-Coefficient* of A and B [Jobson, 1991]. This is the most widely used method to measure correlation. This coefficient, r, is a number between -1 and 1. It is defined by the following four equations:

$$\begin{aligned} r &= s_{AB}/(s_B s_B) \\ s_{AB} &= \sum_{i=1}^{N}(A_i - \mu_A)(B_i - \mu_B)/(N-1) \\ \mu_T &= \sum_{i=1}^{N} T_i/N \\ s_T &= \sqrt{\sum_{i=1}^{N}(T_i - \mu_T)^2/(N-1)}, \quad \text{for } T \in \{ A, B \}. \end{aligned}$$

For a perfect positive correlation $r = 1$, for no correlation $r \approx 0$, and for a perfect negative correlation $r = -1$. Our hypothesis is that $r < 0$.

We will check the hypothesis for the four combinations of density and tightness where the difference in solution time was the most significant and for all random problems. The results are tabulated in Table 5.1. The coefficients in Rows 2, 3 and 4 indicate that there is a clear negative

density	tightness	r
$C = 0.80$	$T = 0.25$	-0.081078
$C = 0.85$	$T = 0.25$	-0.644128
$C = 0.90$	$T = 0.25$	-0.915199
$C = 0.95$	$T = 0.25$	-0.770780
$0 < C < 1$	$0 < T < 1$	-0.699804

Table 5.1: Pearson Product-Moment Correlation-Coefficients for Difference in Solution Time and Difference in Checks

correlation between A and B. It indicates that as A goes up (as MAC-2001 starts spending more time) B goes down (AC-2001 starts saving checks) and vice versa. For $(C, T) = (0.80, 0.25)$ there is a not a significant negative correlation. The "cause" of this insignificant correlation is that there were 5 out of 50 cases where A_i was negative and 2 out of 50 cases where B_i was positive. Furthermore, the absolute value of one of the negative A_i was such that it was second largest of the absolute values of all B_i, $1 \leq i \leq 50$. For the problem sets corresponding to Rows 2, 3 and 4 MAC-2001 always required more time (A_i was always positive) and required fewer checks (B_i was always negative). This is why we should expect a negative coefficient for these problem sets. It is interesting that the coefficient in the last row of Table 5.1 is -0.699804. This indicates that overall there is a significant negative correlation between the difference in

solution time and the difference between the number in checks. It confirms our hypothesis that as soon as MAC-2001 starts spending fewer checks than MAC-3_d it starts to lose out in time.

The last row corresponds to a set of 18050 sample points. This makes the coefficients in Table 5.1 significant.

The Pearson Product-Moment Correlation-Coefficients were computed with dedicated programs written in `C`. To make sure that no errors errors were made these results were verified with `Mathematica`. Computations with `Mathematica` reproduced the result of our own computations up to 6 decimals.

Figure 5.27 depicts a scatter plot of the difference in checks against the difference in time. There are only a few combinations for which MAC-2001 was faster. The large cluster located at the top right hand side has a "slope" that corresponds to the negative correlation and a position that re-confirms that MAC-3_d was better in wall time

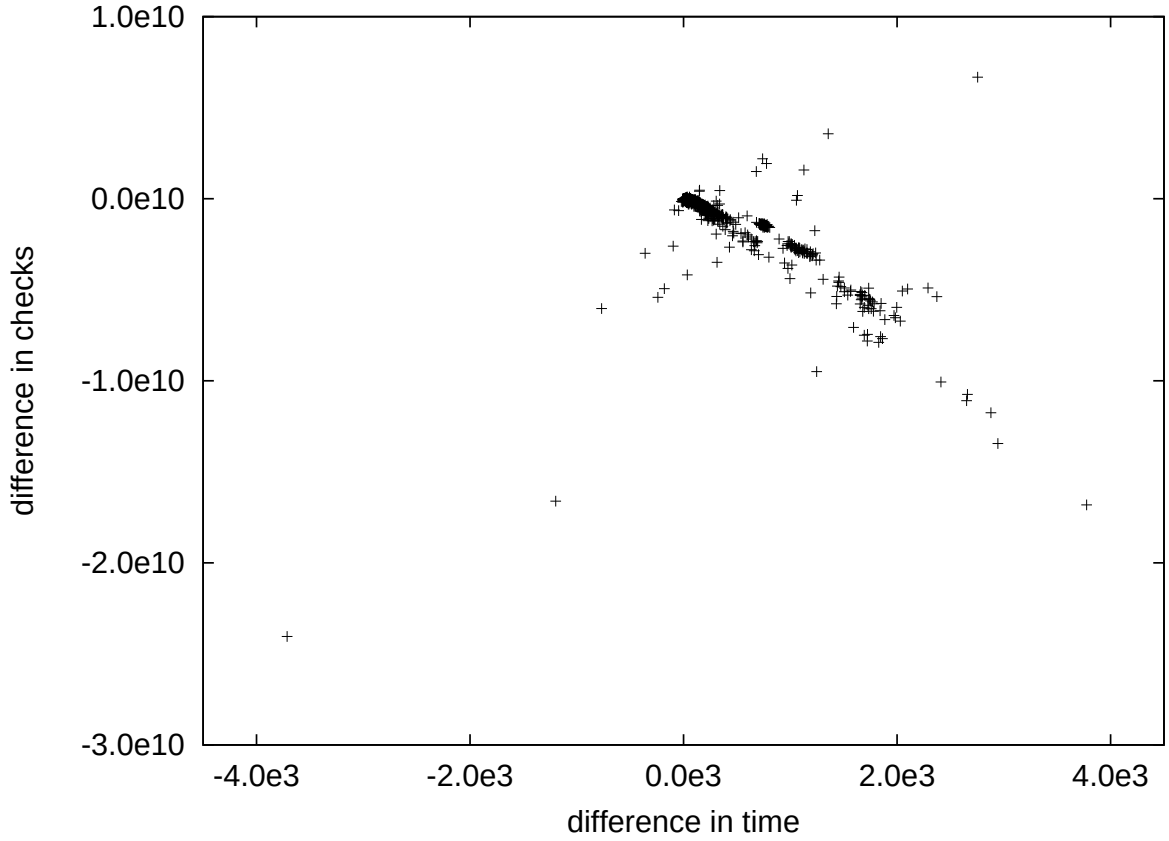

Figure 5.27: $n = 30$, $d = 30$, Search: Scatter plot, difference in checks against difference in time.

Chapter 6

Conclusions and Recommendations

We have compared two arc-consistency algorithms called AC-2001 and AC-3_d. AC-2001 is optimal in its worst case time-complexity, whereas AC-3_d is not. We have compared both algorithms for stand alone arc-consistency and for MAC search. The results from the comparison demonstrate that, despite common belief to the contrary, for an arc-consistency algorithm to be useful during search there is no need for it to have an optimal worst case time-complexity. We have presented results that suggest quite the opposite; To avoid duplication of checks in MAC search may be harmful. This claim is supported by the observation that MAC-3_d was never significantly slower than MAC-2001, that MAC-3_d was usually significantly faster and, most importantly, that as soon as MAC-2001 started to become successful in avoiding the duplication of many checks it started to lose much in time. However, more work has to be done to properly support this claim.

Our results indicate that AC-3_d is a good substitute for AC-2001 for stand alone arc-consistency. When it comes to completing its task fast AC-3_d is a clear winner. When it comes to saving checks there is no clear overall winner. If tightness is about 0.85 then AC-2001 is the preferred algorithm for minimising checks. Otherwise AC-3_d is the preferred choice.

Any arc-consistency algorithm that is the basis of a general purpose MAC solver should be good *on average*. Yet, most research on arc-consistency algorithms is focusing on the design of algorithms that are optimal in *worst case* scenarios, insisting that *at any cost* these cases should be dealt with as efficiently as possible. This report has provided evidence that to insist on being able to deal efficiently with worst case scenarios may not be the best strategy when it comes to solving quickly. The holy grail of arc-consistency research is to design arc-consistency algorithms that are optimal in their *average* time-complexity. AC-3_d's clever use of arc-heuristics and domain-heuristics seems to improve this average time-complexity. A more detailed study on the impact of these heuristics may provide us with useful insights on how to design new and better arc-consistency algorithms. These insights may also shed light on how to design other efficient consistency algorithms.

Acknowledgements

I should like to thank Christian Bessière for the use of his solver, for help with its installment, and for early discussions. Also I wish to thank Christian van den Bosch for setting up and carrying out the experiments. Many thanks should also go to Rick Wallace for his suggestion to use the Pearson Product-Moment Correlation-Coefficient and for useful discussions. Finally, I should like to express my gratitude to Gene Freuder for his support of this work. This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075.

Bibliography

[Bessière and Régin, 2001] C. Bessière and J.-C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'2001)*, pages 309–315, 2001.

[Bessière *et al.*, 1995] C. Bessière, E.C. Freuder, and J.-C. Régin. Using inference to reduce arc consistency computation. In C.S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, volume 1, pages 592–598, Montréal, Québec, Canada, 1995. Morgan Kaufmann Publishers, Inc., San Mateo, California, USA.

[Bessière *et al.*, 1999] C. Bessière, E.G. Freuder, and J.-C. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1):125–148, 1999.

[Collart *et al.*, 1997] S. Collart, M. Kalkbrener, and D. Mall. Converting bases with the Gröbner walk. *Journal of Symbolic Computation*, 24(3 and 4):465–470, 1997.

[Gaschnig, 1978] J. Gaschnig. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems. In *Proceeding of the Second Biennial Conference, Canadian Society for the Computational Studies of Intelligence*, pages 268–277, 1978.

[Gent *et al.*, 1996] I.P. Gent, MacIntyre E., P. Prosser, B.M. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In E.C. Freuder, editor, *Principles and Practice of Constraint Programming*, pages 179–193. Springer, 1996.

[Gent *et al.*, 1997] I.P. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh. The constrainedness of arc consistency. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'1997)*, pages 327–340. Springer, 1997.

[Gent *et al.*, 2001] Ian Gent, Ewan MacIntyre, Patrick Prosser, Barbara Smith, and Toby Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.

[Jobson, 1991] J.D. Jobson. *Applied Multivariate Data Analysis*, volume I: Regression and Experimental Design. Springer, 1991.

[Mackworth and Freuder, 1985] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–73, 1985.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mohr and Henderson, 1986] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Sabin and Freuder, 1994] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A.G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 125–129. John Wiley & Sons, 1994.

[van Dongen, 2002a] M.R.C. van Dongen. *Constraints, Varieties, and Algorithms*. PhD thesis, Department of Computer Science, University College Cork, Ireland, 2002.

[van Dongen, 2002b] M.R.C. van Dongen. Domain-heuristics for arc-consistency algorithms. In K.R. Apt, F. Fages, E.G. Freuder, B. O'Sullivan, F. Rossi, and T. Walsh, editors, *ERCIM/Colognet Workshop, Cork*, pages 72–83, 2002.

[van Dongen, 2002c] M.R.C. van Dongen. AC-3_d an efficient arc-consistency algorithm with a low space-complexity. In P. Van Hentenryck, editor, *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, pages 755–760. Springer, 2002.

[van Dongen, 2002d] M.R.C. van Dongen. AC-3_d an efficient arc-consistency algorithm with a low space-complexity. Technical Report TR-01-2002, Cork Constraint Computation Centre/CS Department UCC, 2002.

[Wallace and Freuder, 1992] R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *AI/GI/VI '92*, pages 163–169, Vancouver, British Columbia, Canada, 1992.

[Wallace, 1993] R.J. Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In R. Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 239–245, 1993.

[Zhang and Yap, 2001] Y. Zhang and R.H.C. Yap. Making AC-3 an optimal algorithm. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'2001)*, pages 316–321, 2001.

Appendix A

Graphs

This chapter contains graphs for all the experiments carried out. The data were obtained from experiments that were set up as described in Chapter 5. Sections are organised by the number of variables n and domain size d.

A.1 Results for $N = D = 20$

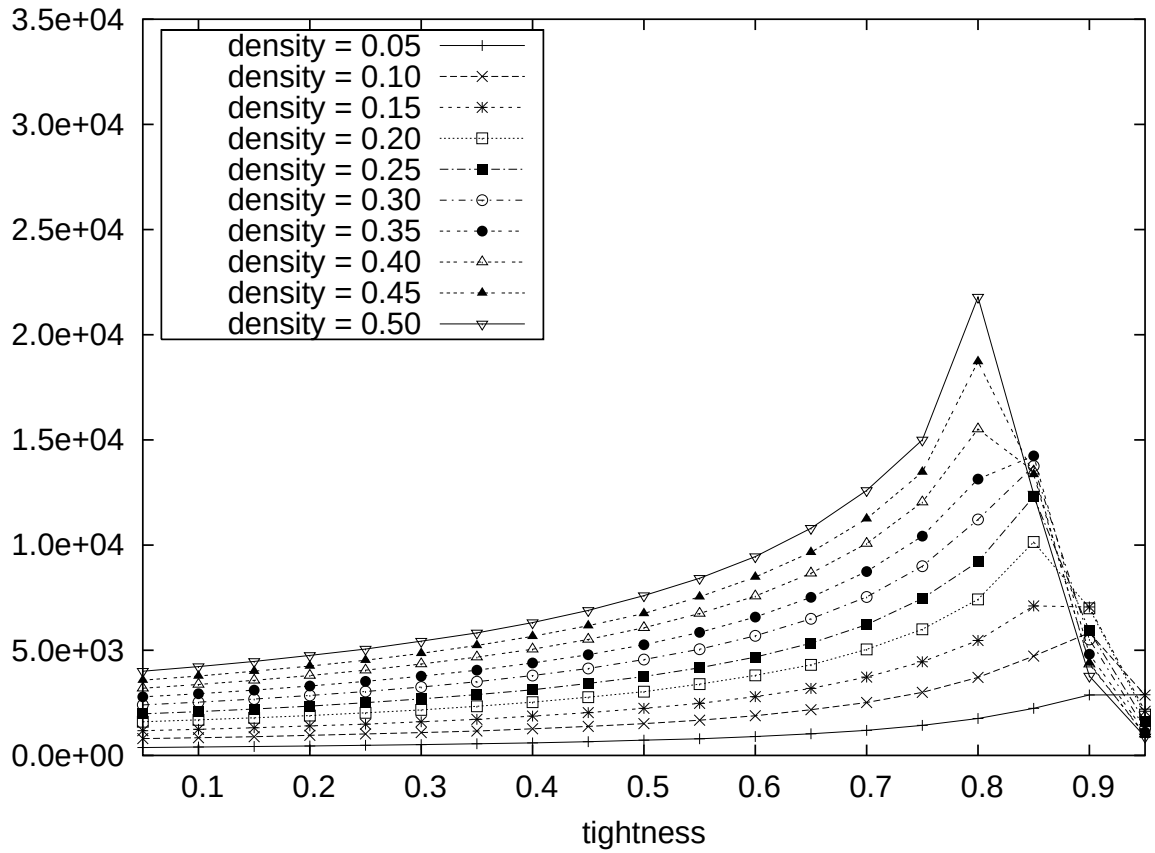

Figure A.1: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001.

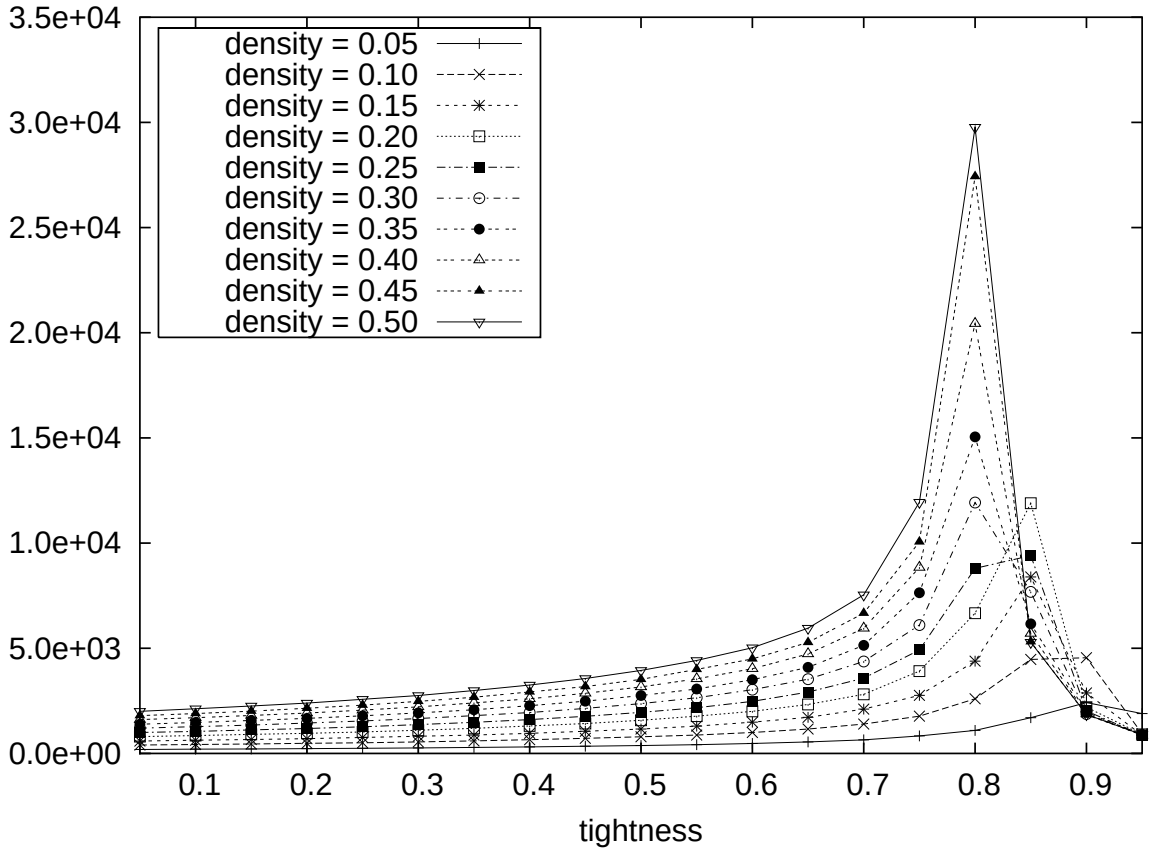

Figure A.2: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-3_d.

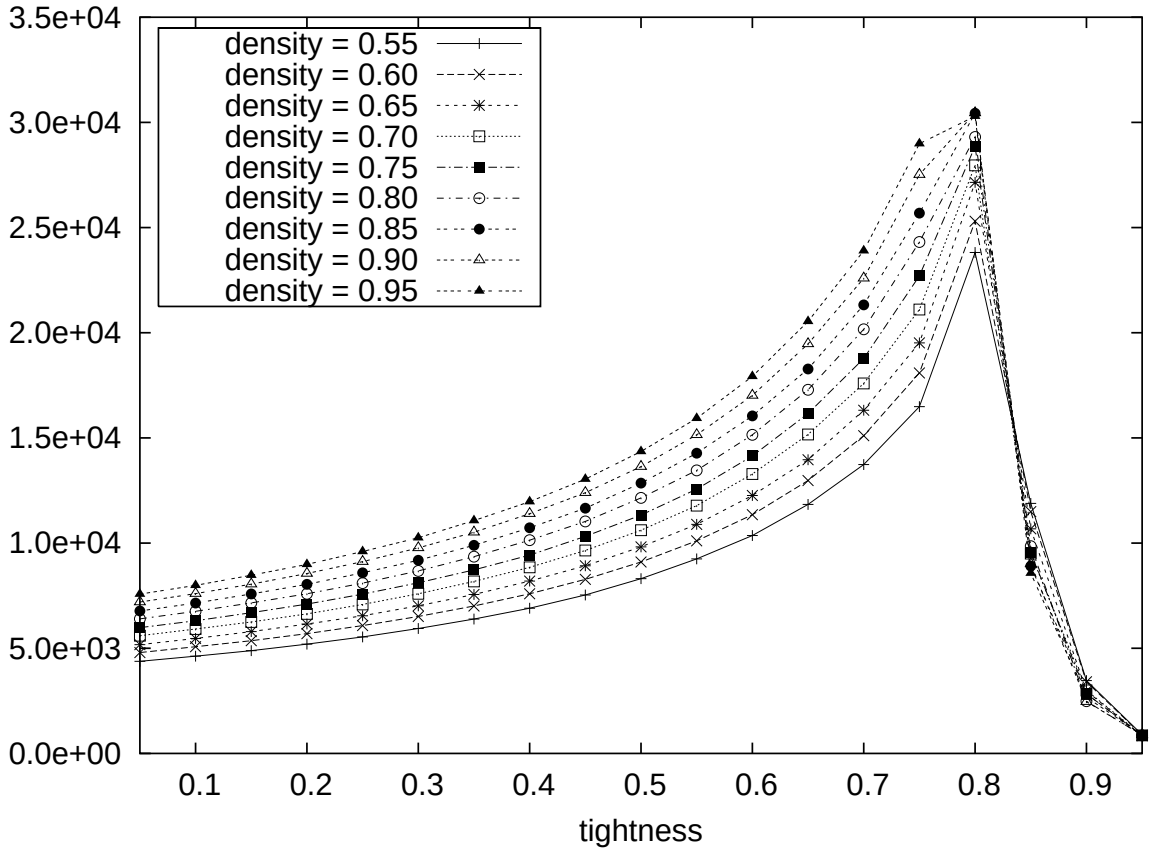

Figure A.3: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-2001.

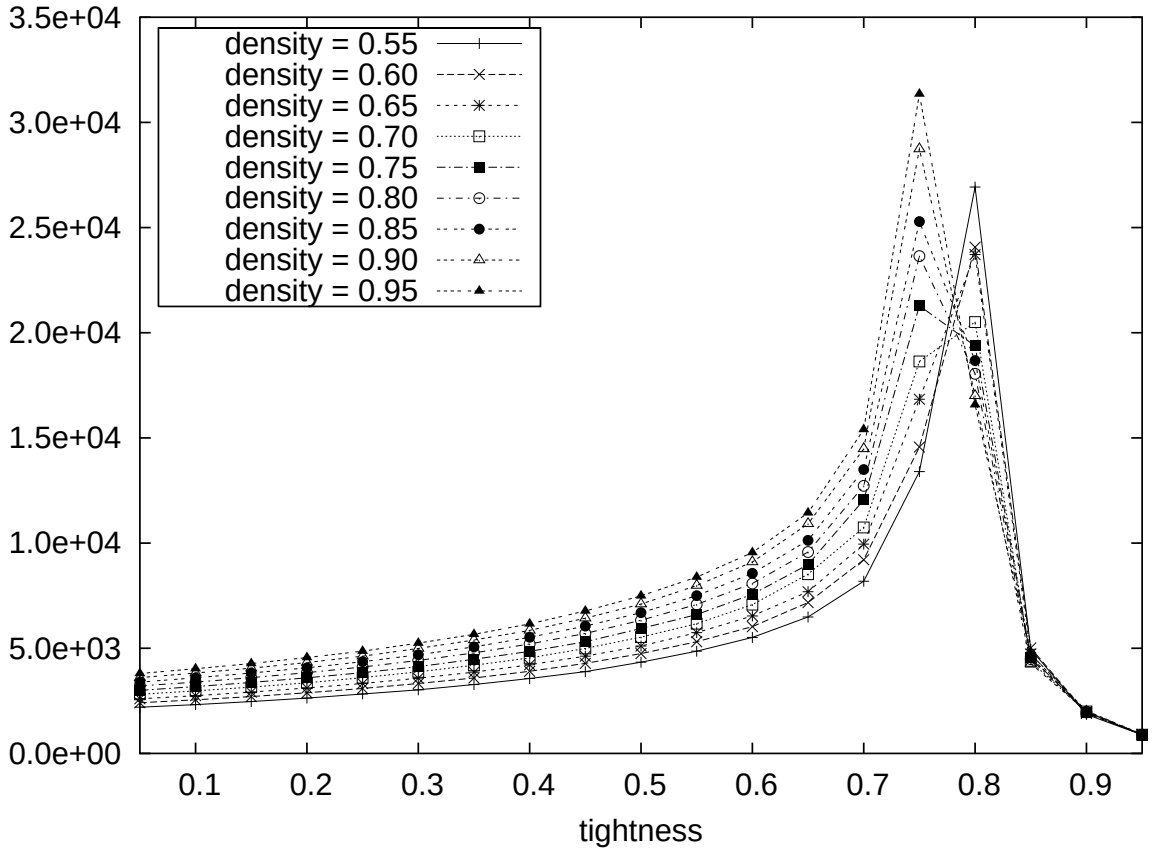

Figure A.4: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-3_d.

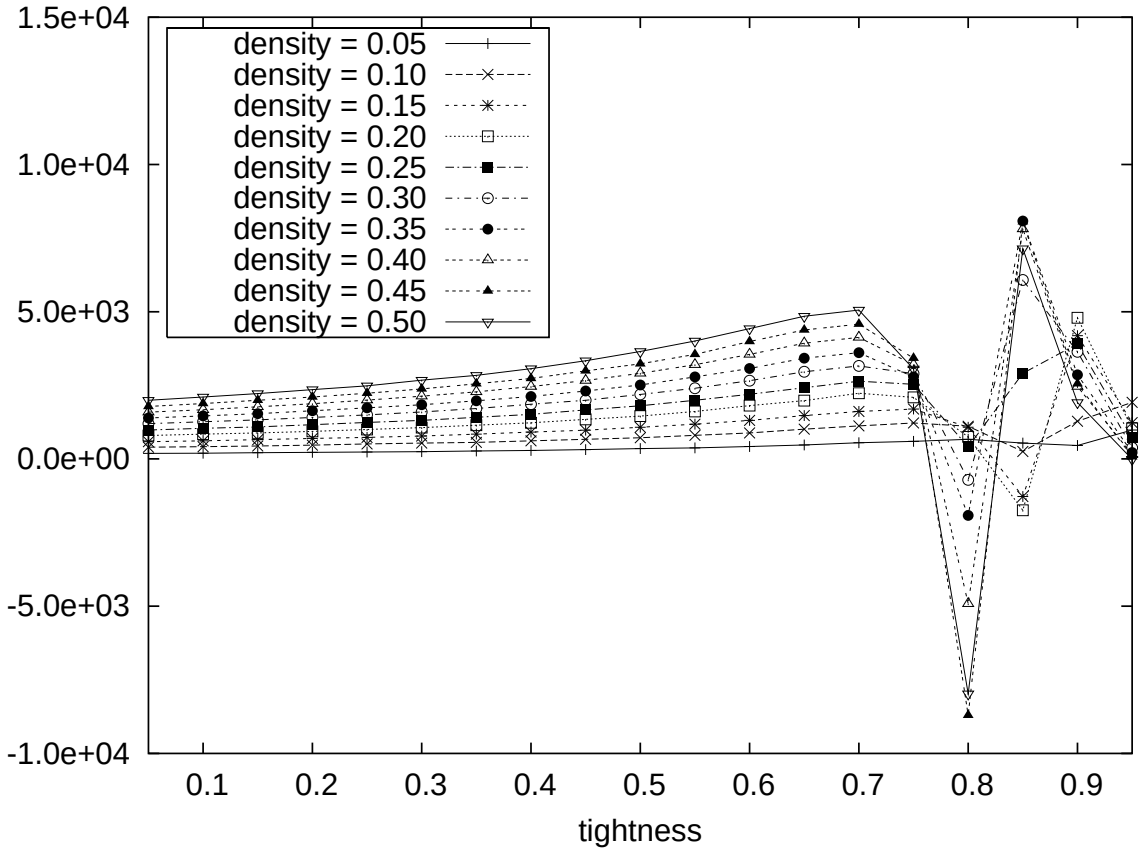

Figure A.5: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001 $-$ AC-3$_d$.

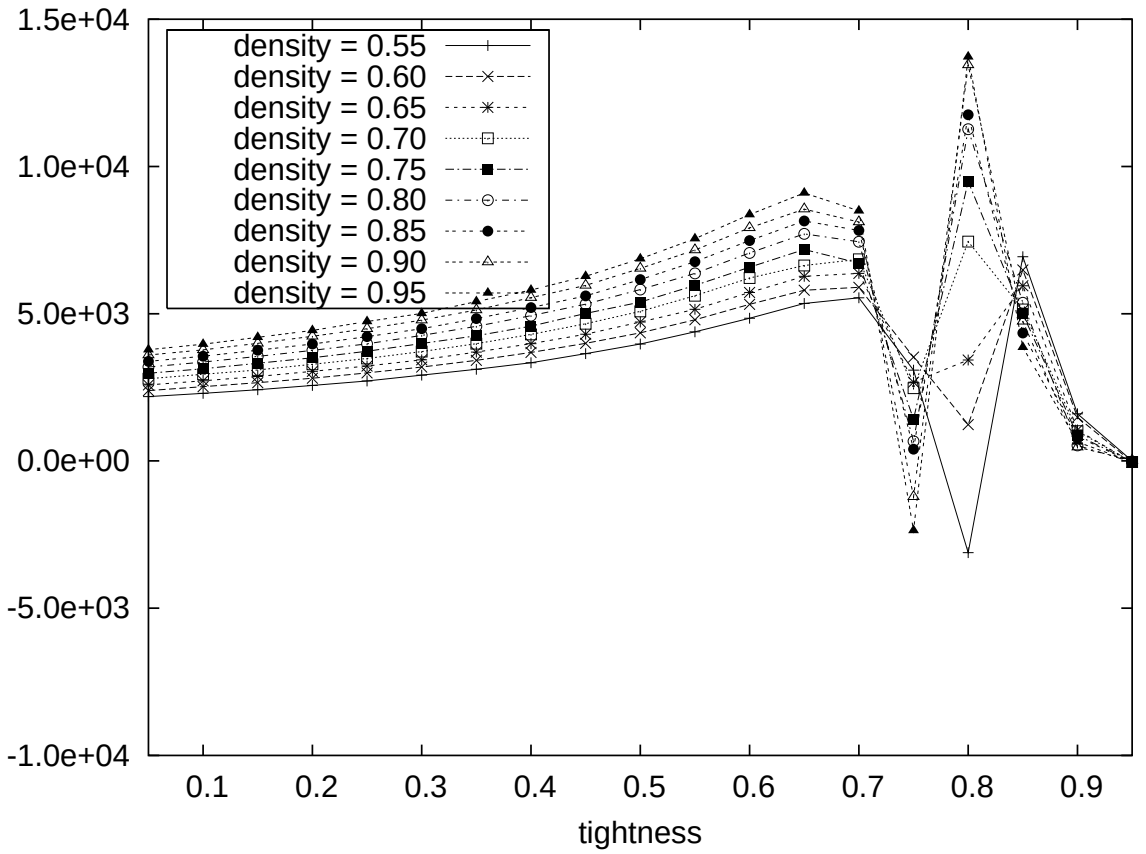

Figure A.6: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-2001 $-$ AC-3$_d$.

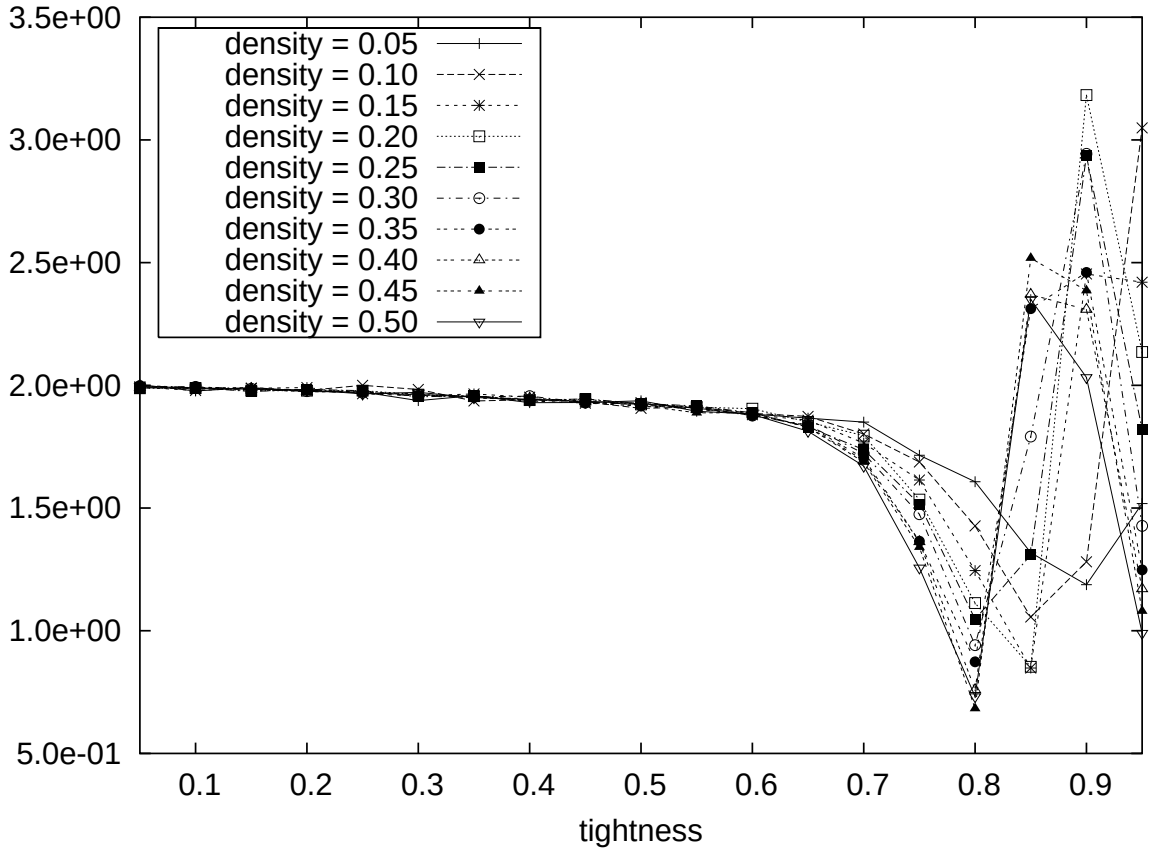

Figure A.7: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001/AC-3$_d$.

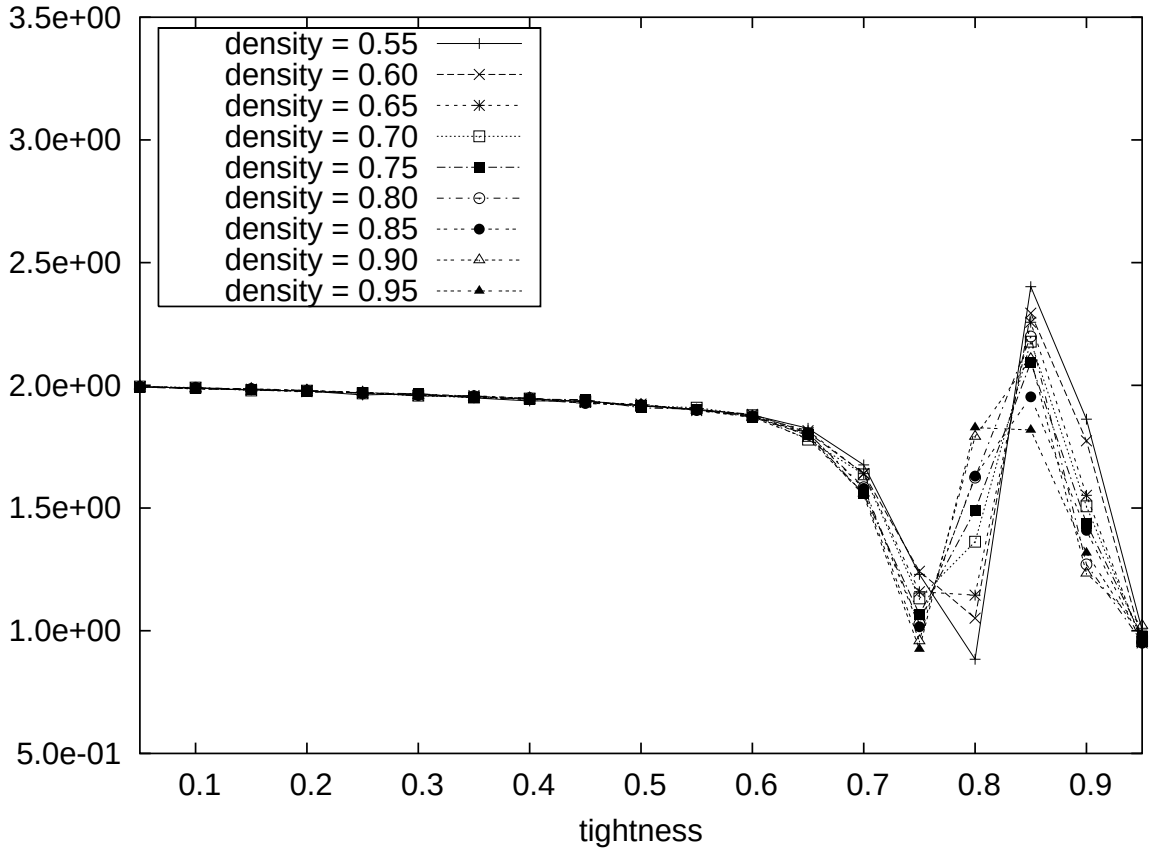

Figure A.8: $n = 20$, $d = 20$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-2001/AC-3_d.

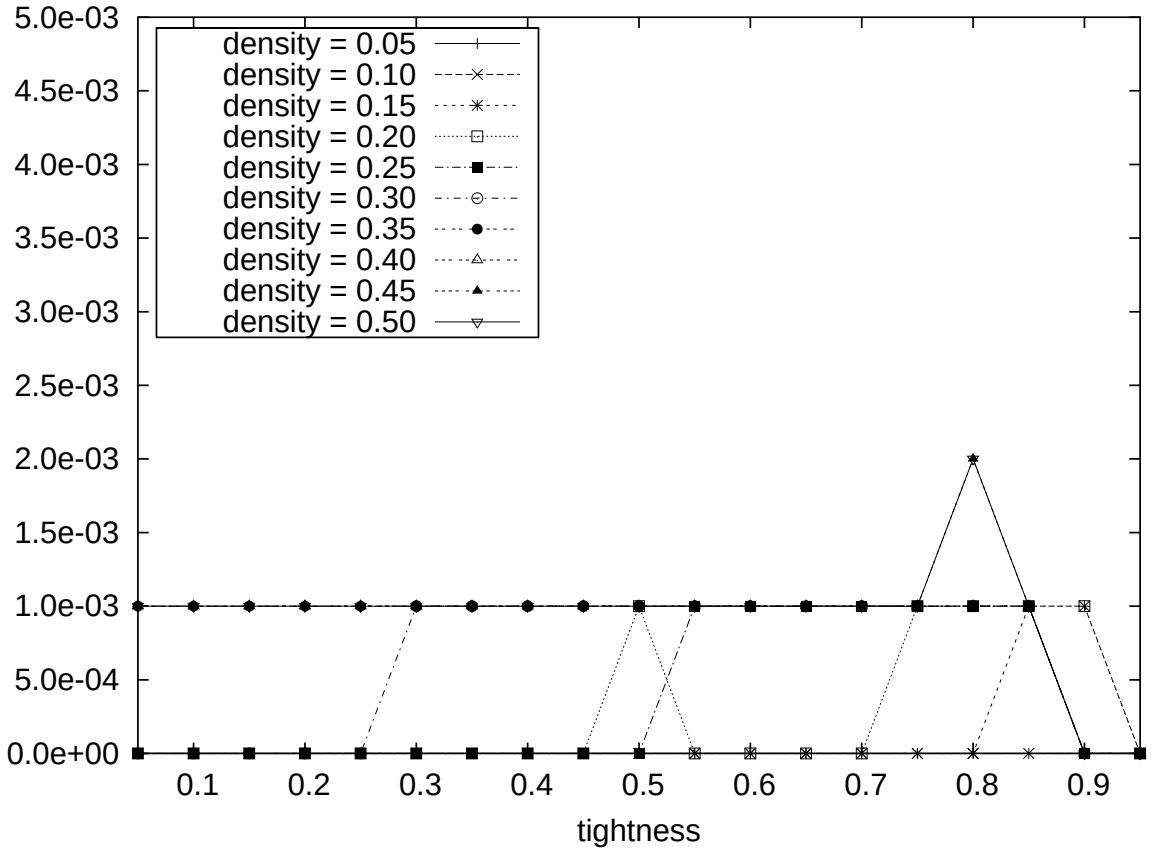

Figure A.9: $n = 20$, $d = 20$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-2001.

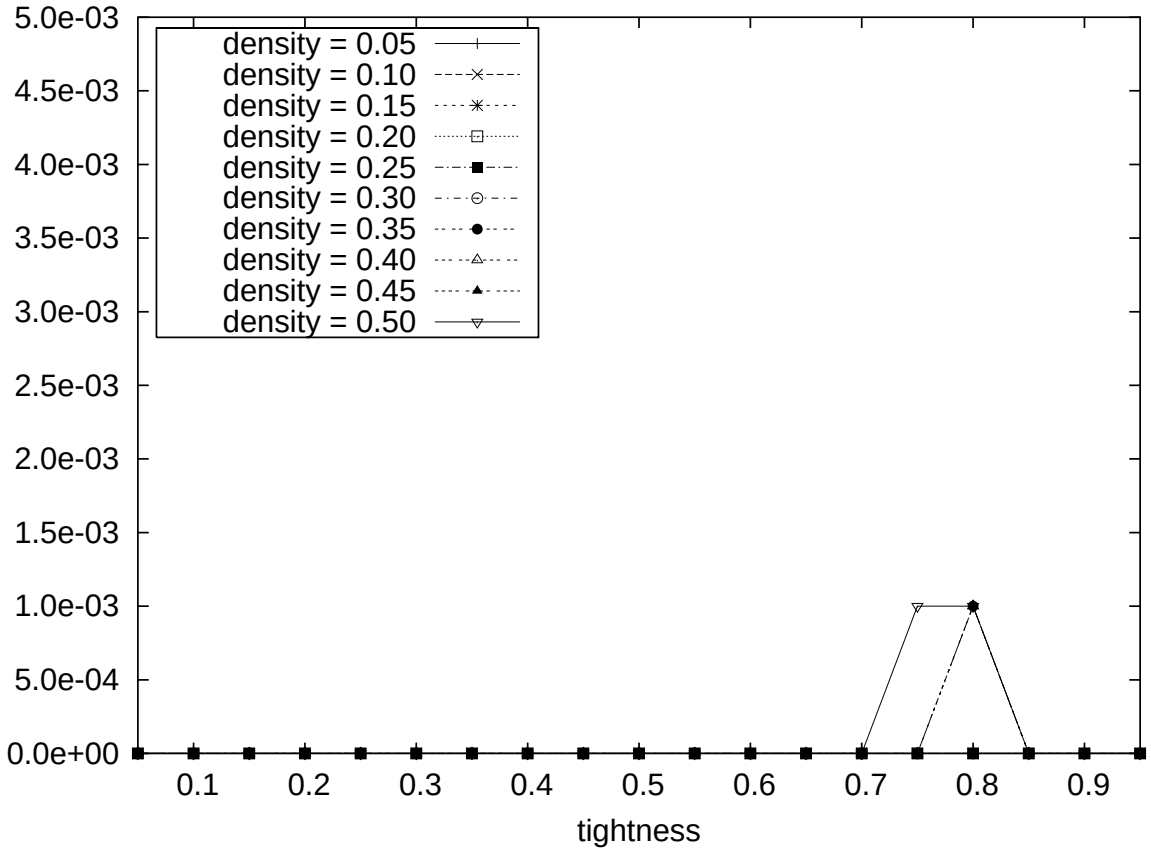

Figure A.10: $n = 20$, $d = 20$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-3_d.

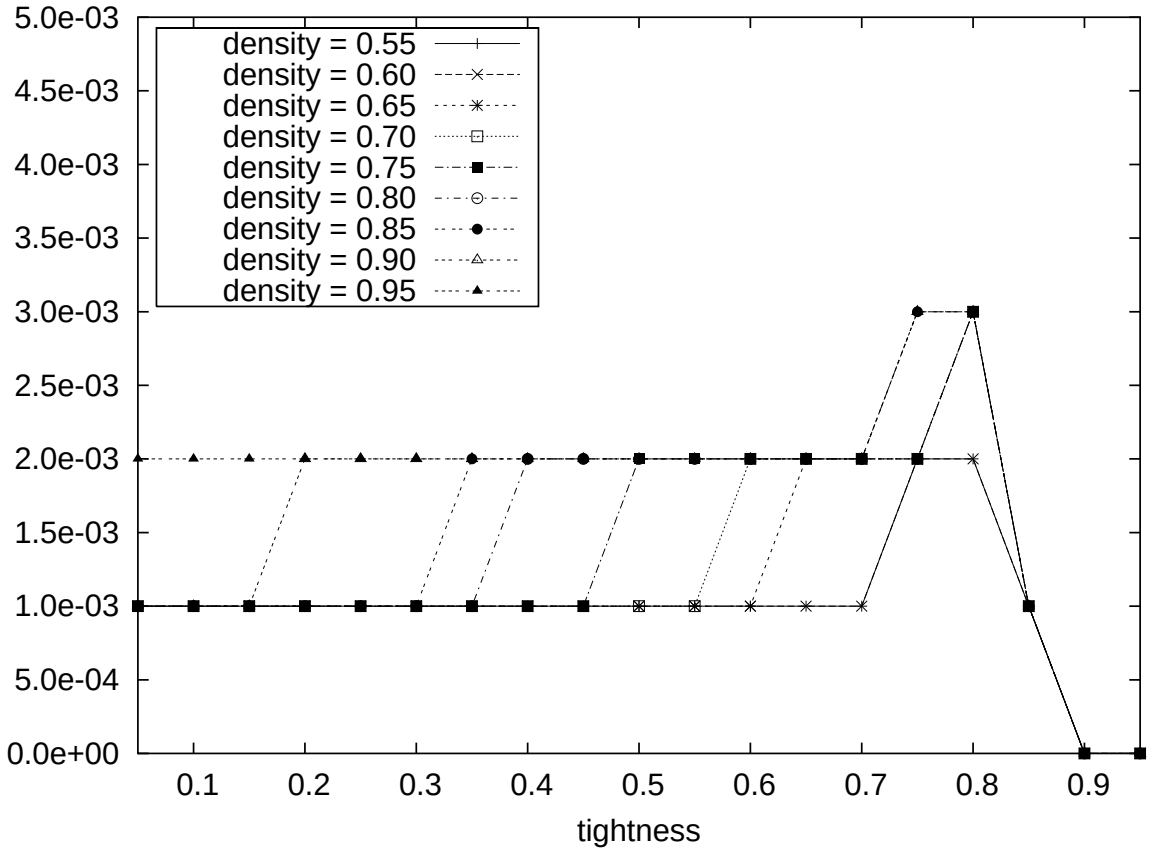

Figure A.11: $n = 20$, $d = 20$, Stand alone arc-consistency: Time, $C > 0.5$, AC-2001.

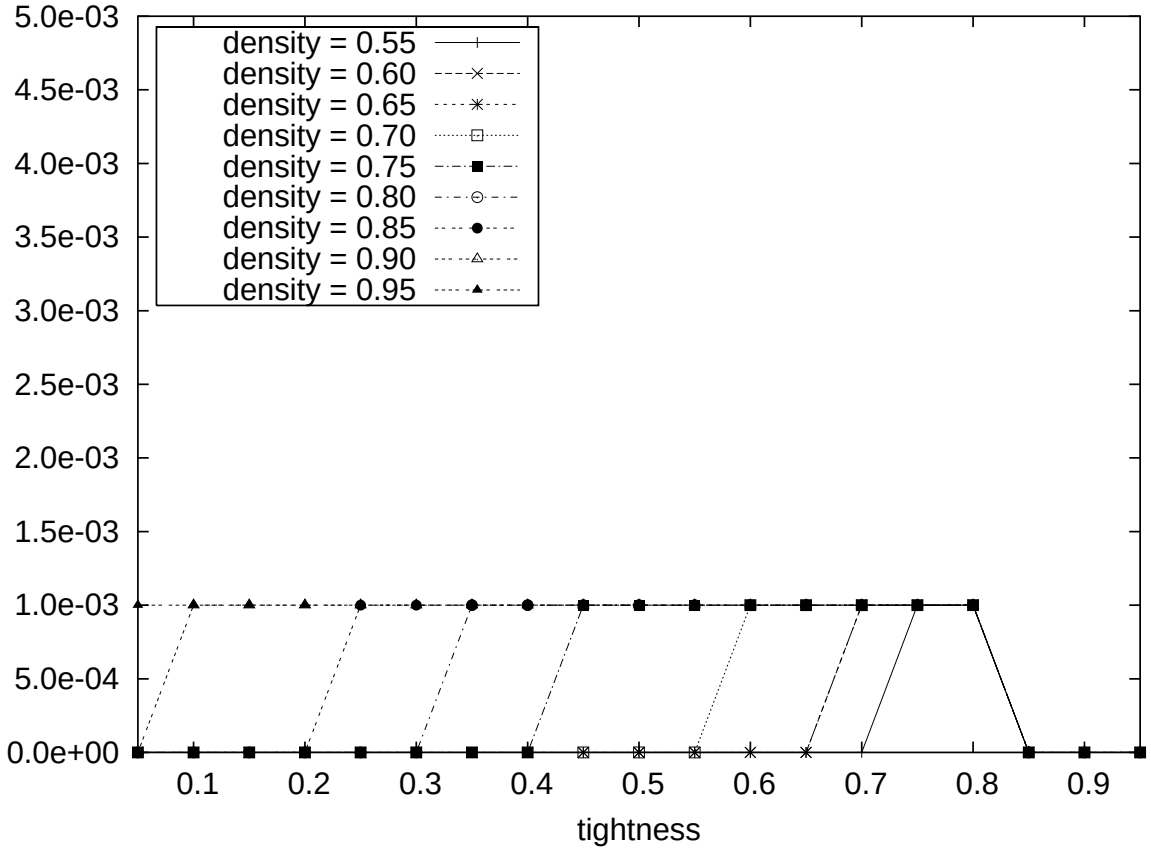

Figure A.12: $n = 20$, $d = 20$, Stand alone arc-consistency: Time, $C > 0.5$, AC-3_d.

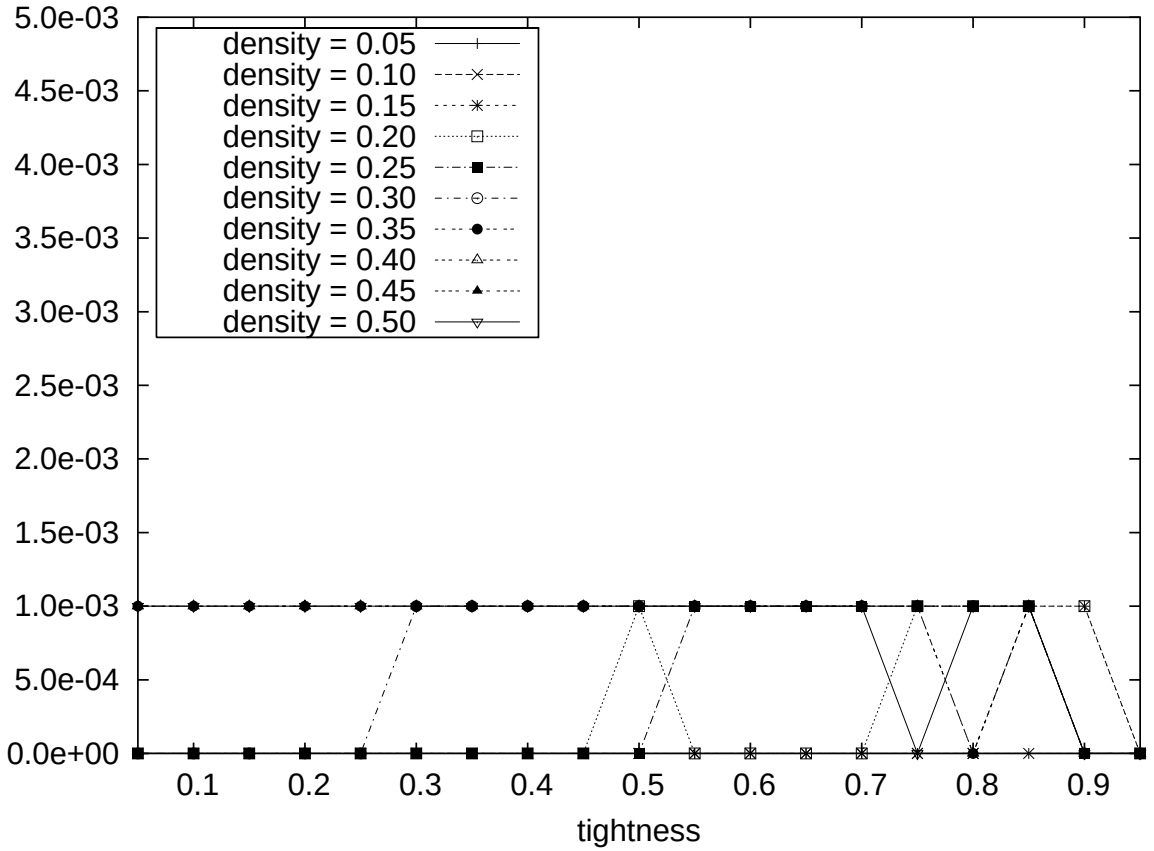

Figure A.13: $n = 20$, $d = 20$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-2001 – AC-3_d.

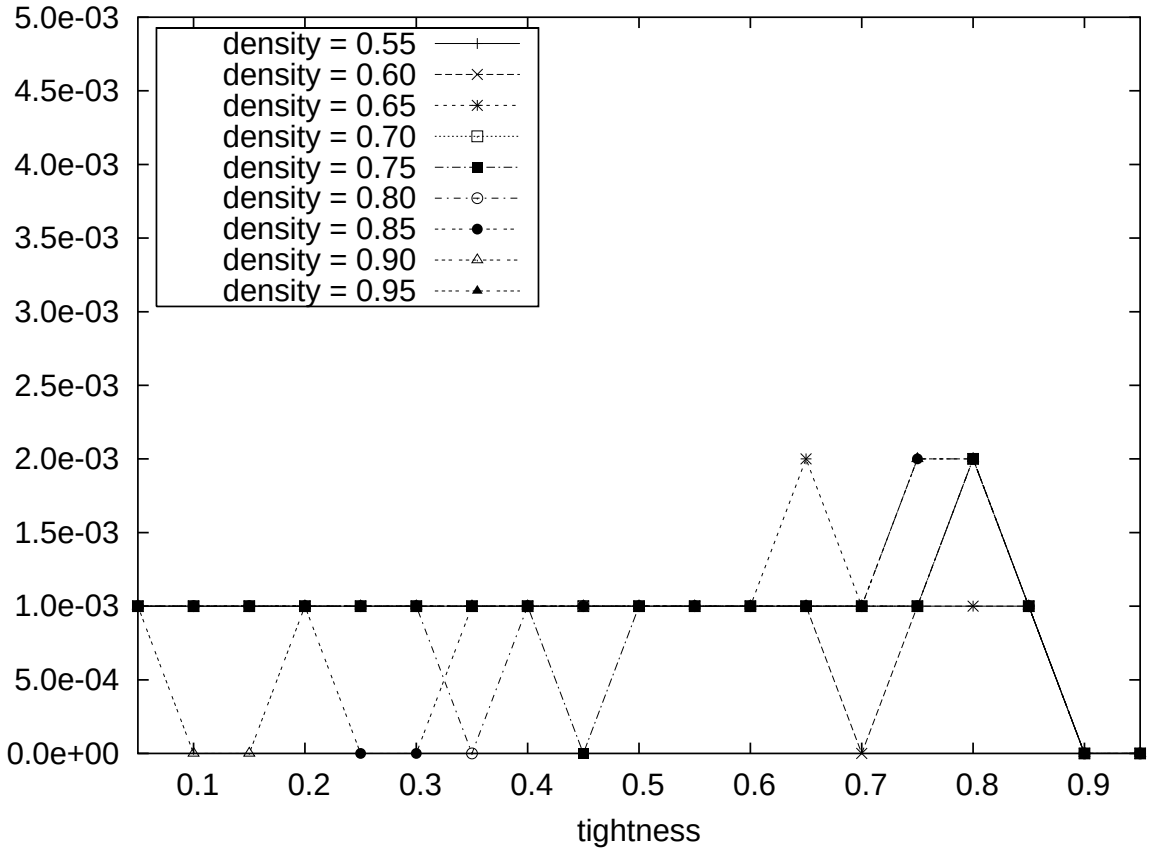

Figure A.14: $n = 20$, $d = 20$, Stand alone arc-consistency: Time, $C > 0.5$, AC-2001 – AC-3_d.

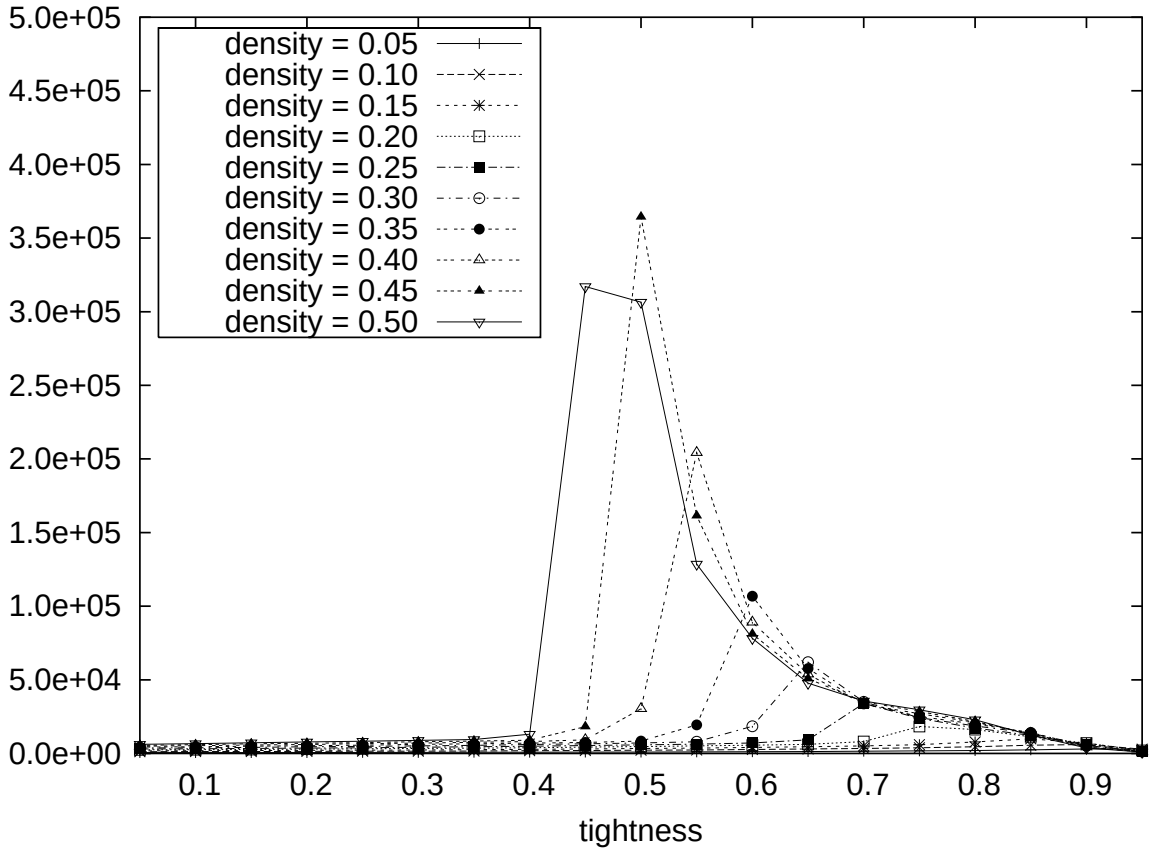

Figure A.15: $n = 20$, $d = 20$, Search: Checks, $C \leq 0.5$, AC-2001.

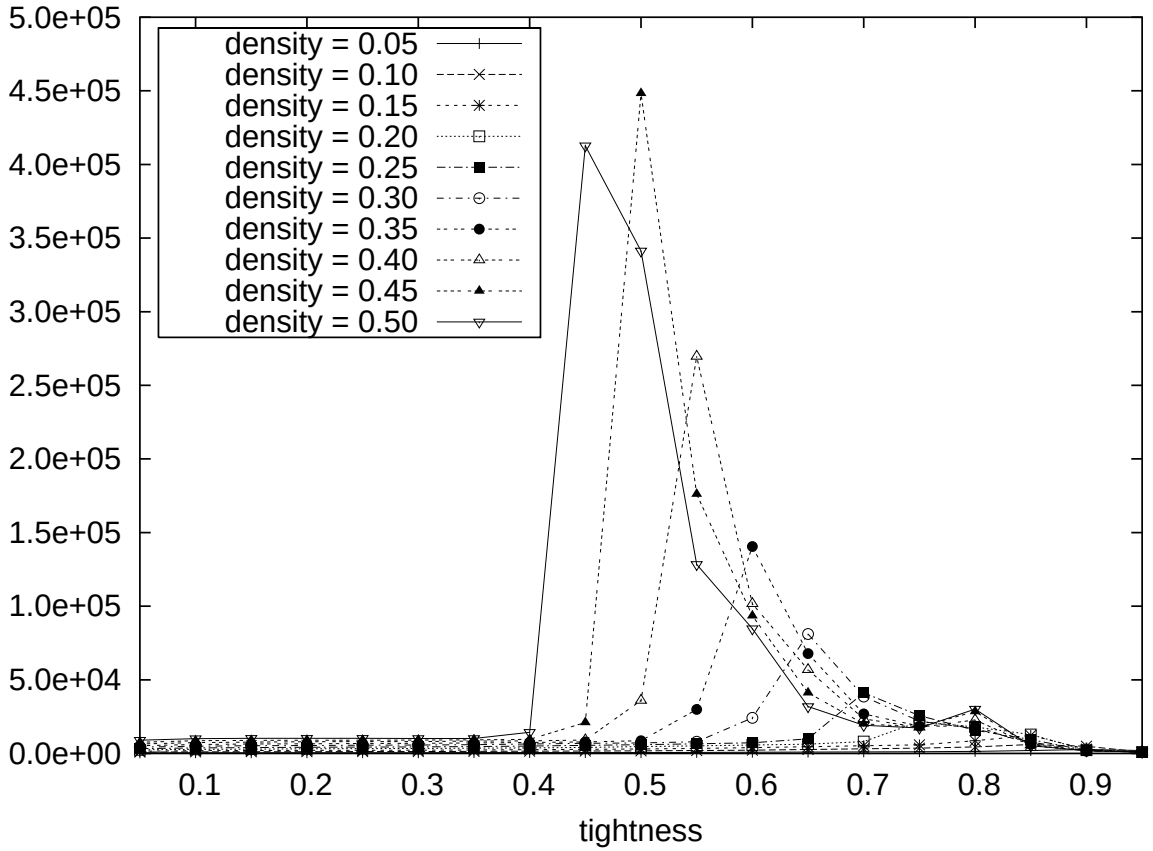

Figure A.16: $n = 20$, $d = 20$, Search: Checks, $C \leq 0.5$, AC-3_d.

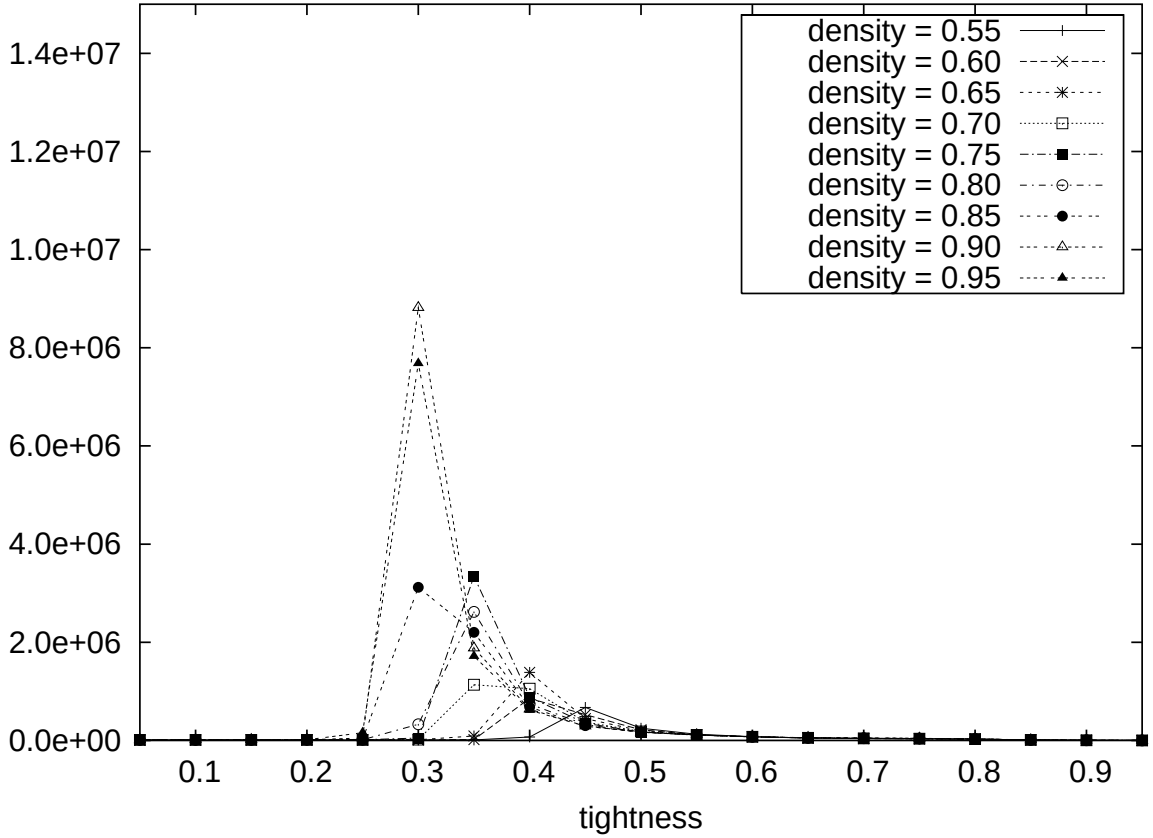

Figure A.17: $n = 20$, $d = 20$, Search: Checks, $C > 0.5$, AC-2001.

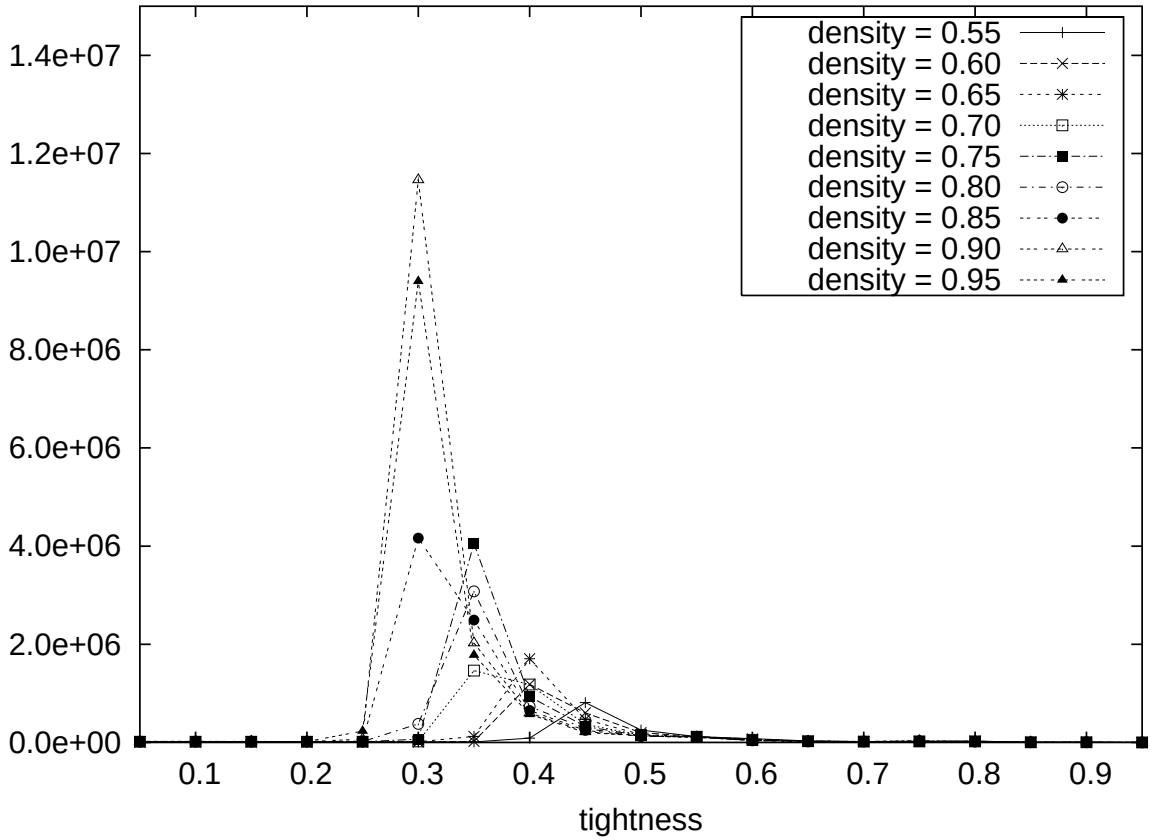

Figure A.18: $n = 20$, $d = 20$, Search: Checks, $C > 0.5$, AC-3_d.

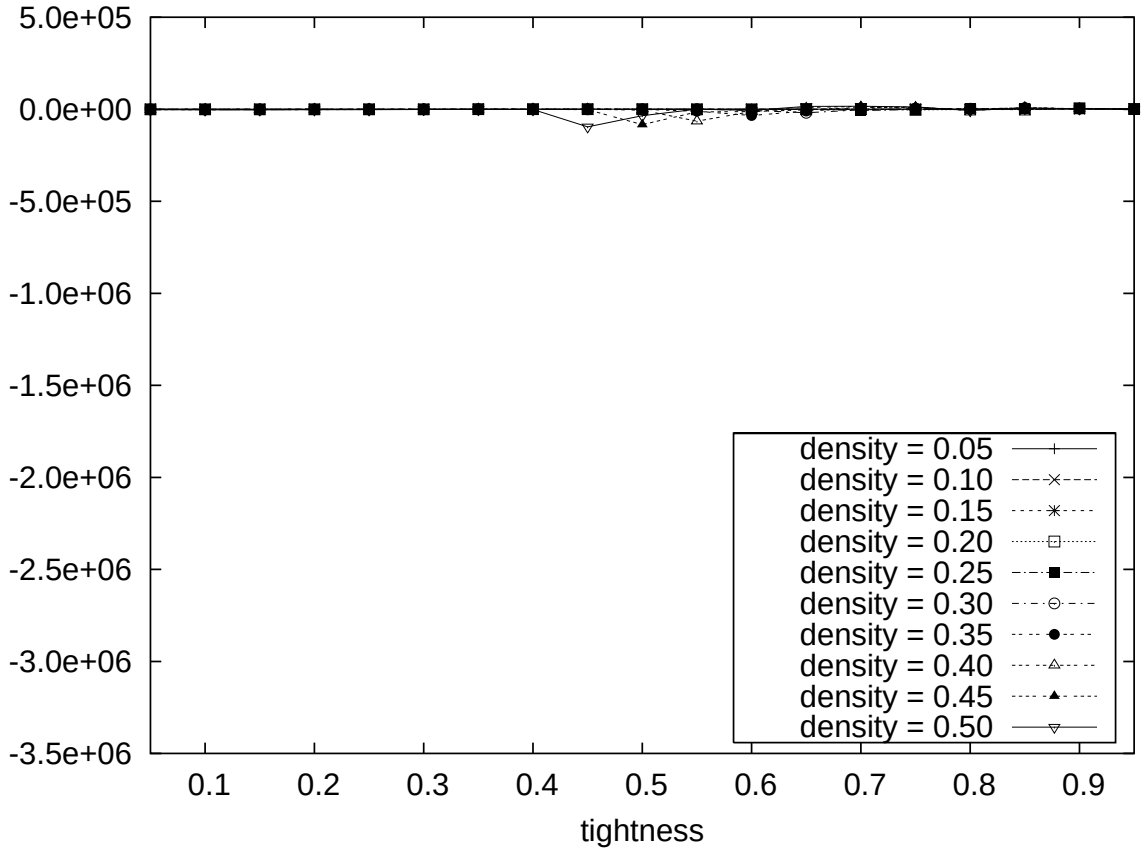

Figure A.19: $n = 20$, $d = 20$, Search: Checks, $C \leq 0.5$, AC-2001 $-$ AC-3$_d$.

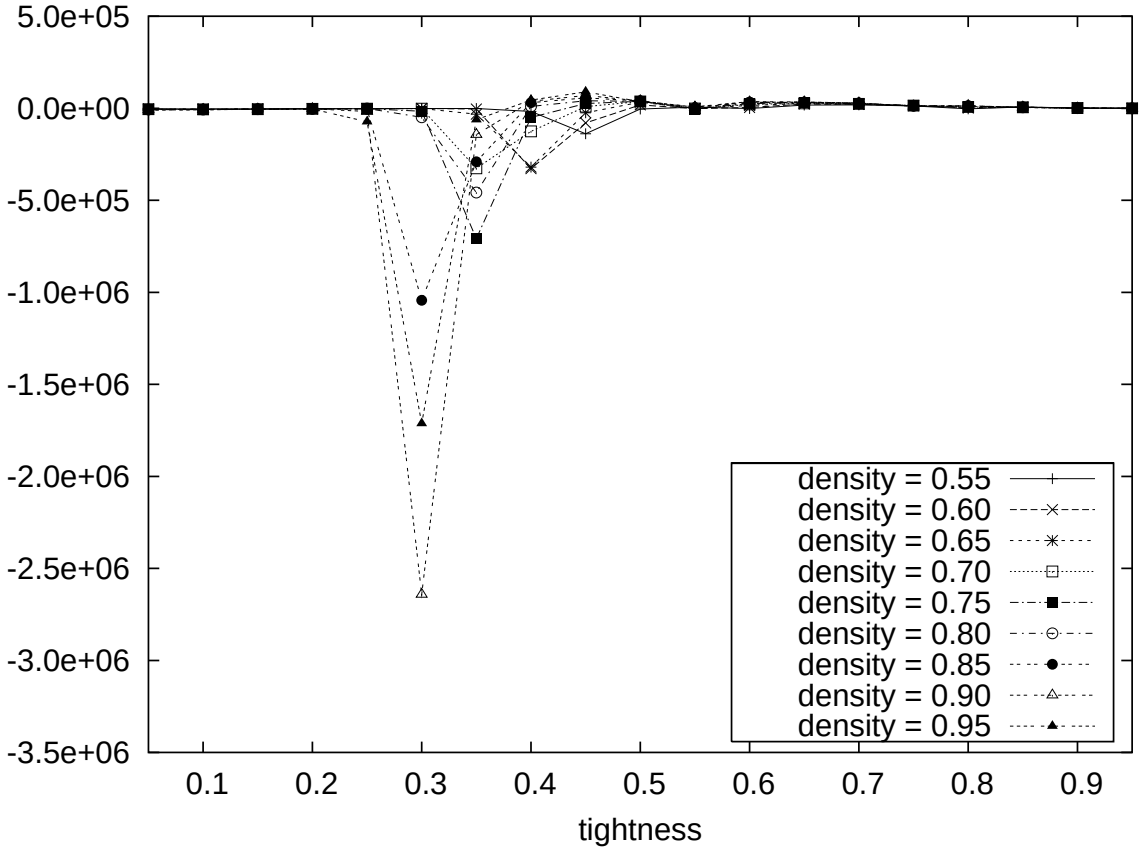

Figure A.20: $n = 20$, $d = 20$, Search: Checks, $C > 0.5$, AC-2001 $-$ AC-3$_d$.

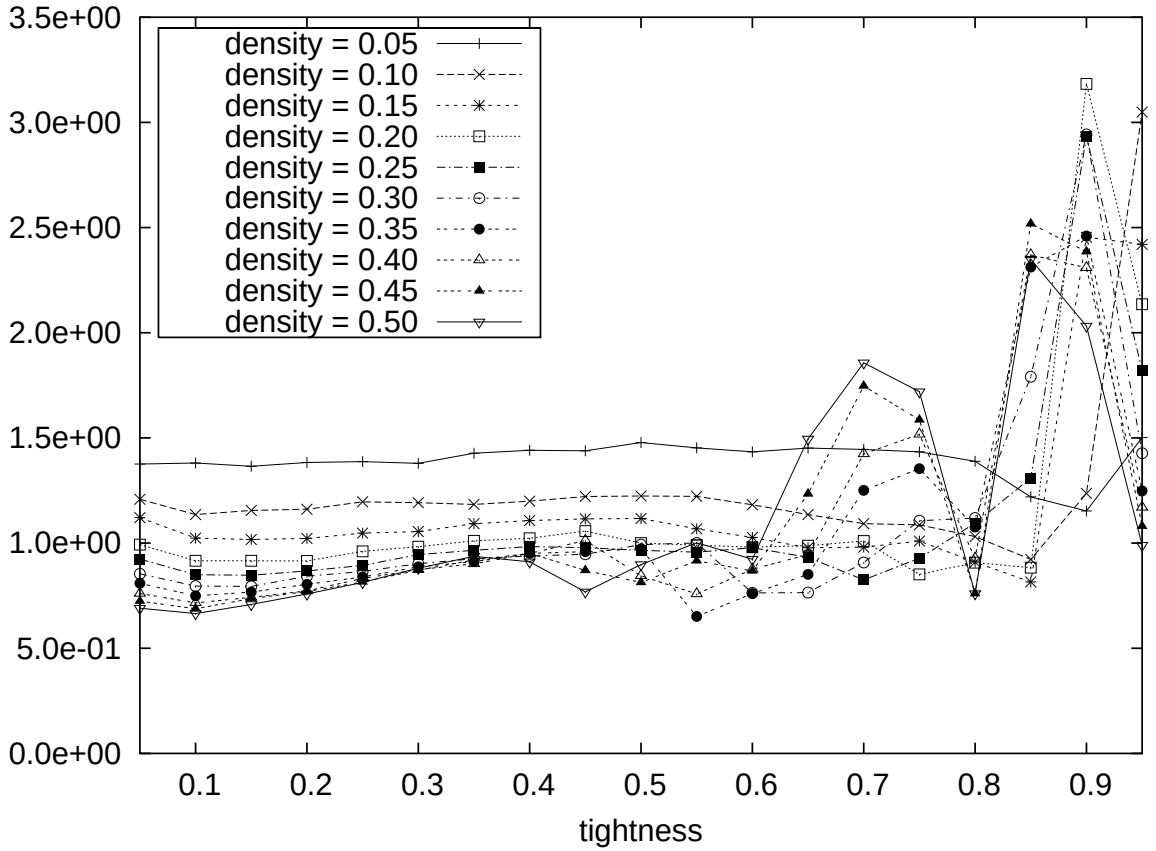

Figure A.21: $n = 20$, $d = 20$, Search: Checks, $C \leq 0.5$, AC-2001/AC-3_d.

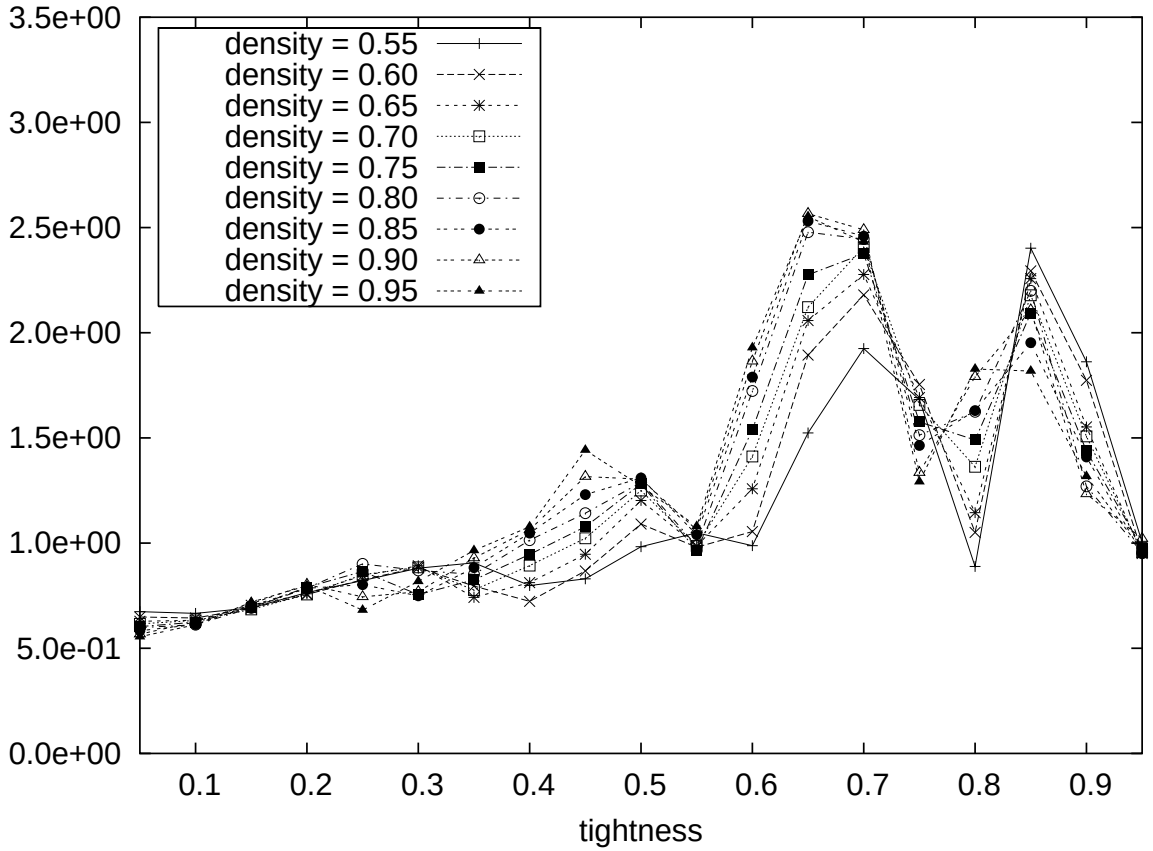

Figure A.22: $n = 20$, $d = 20$, Search: Checks, $C > 0.5$, AC-2001/AC-3$_d$.

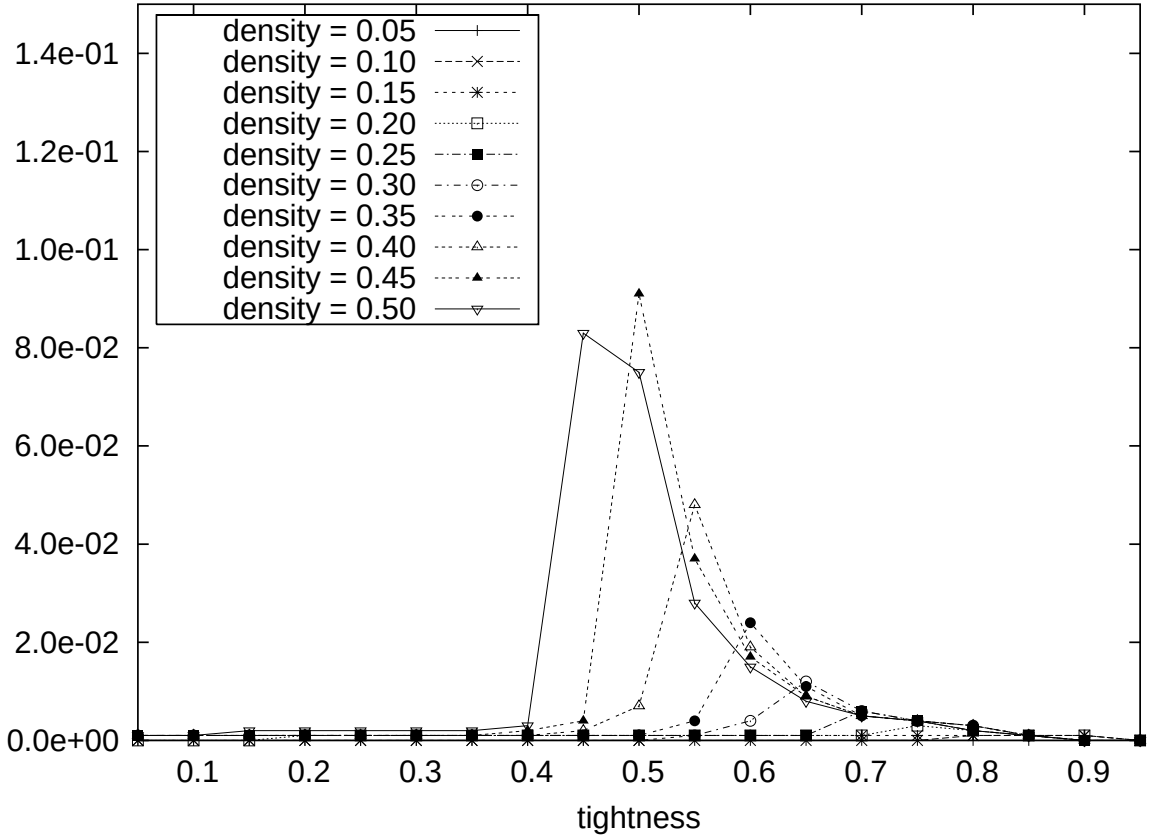

Figure A.23: $n = 20$, $d = 20$, Search: Time, $C \leq 0.5$, AC-2001.

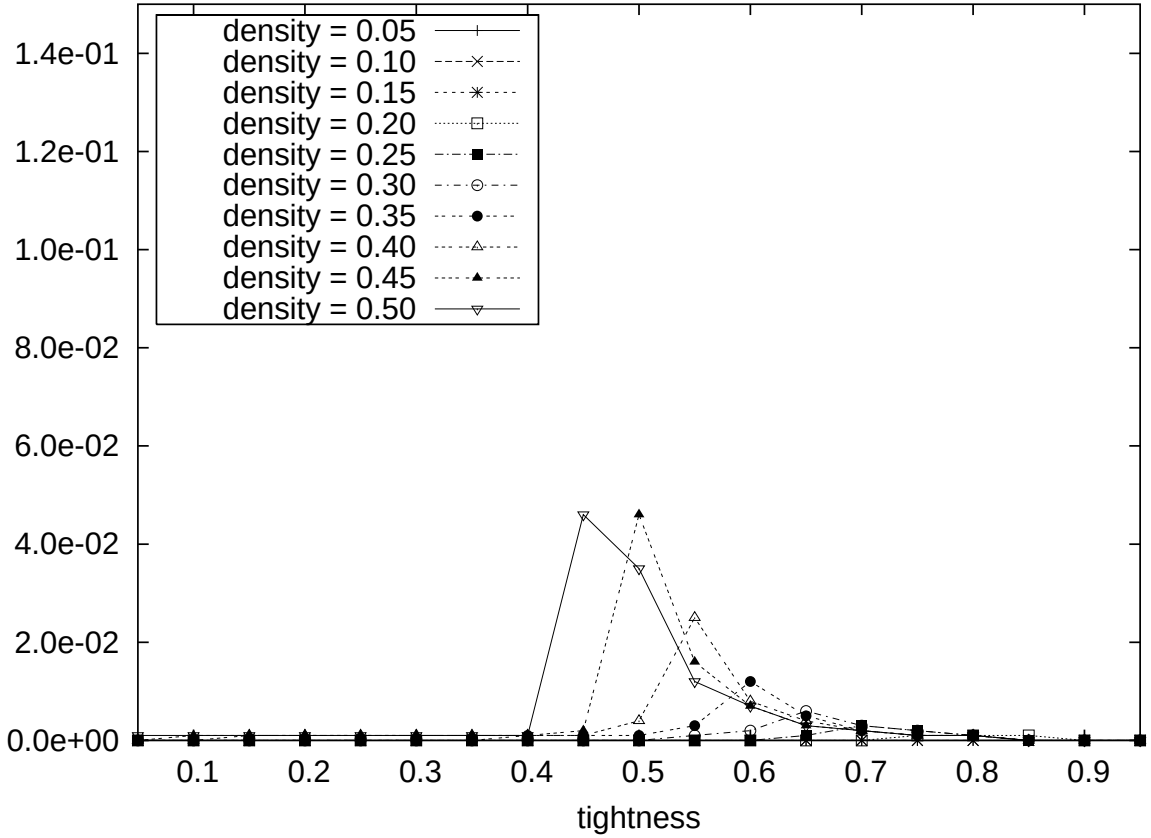

Figure A.24: $n = 20$, $d = 20$, Search: Time, $C \leq 0.5$, AC-3_d.

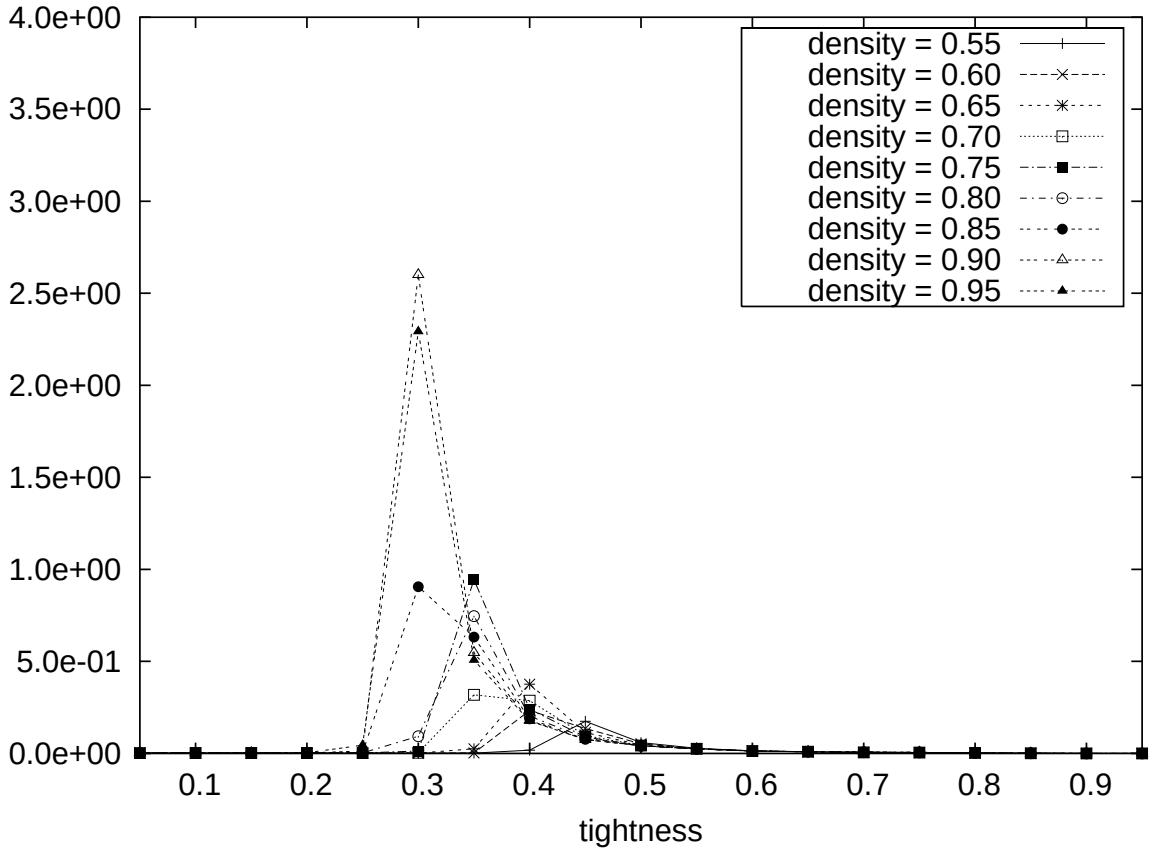

Figure A.25: $n = 20$, $d = 20$, Search: Time, $C > 0.5$, AC-2001.

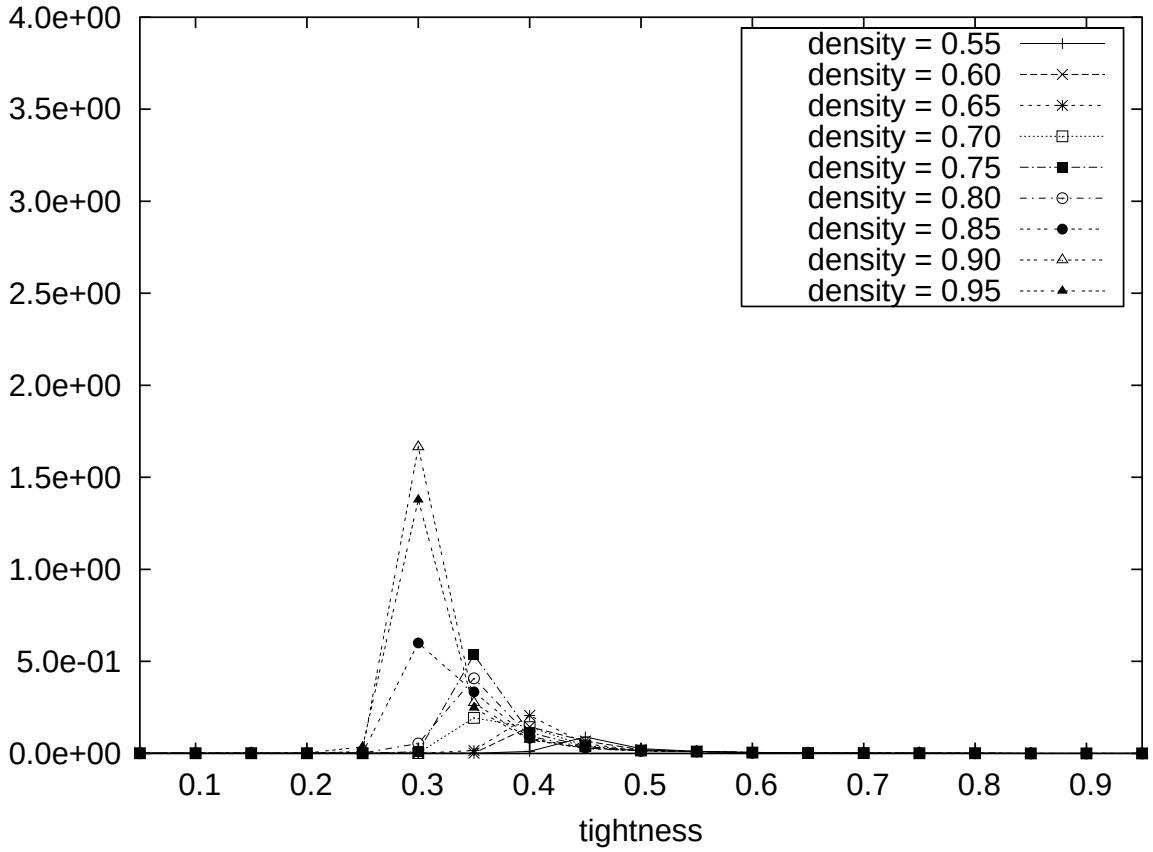

Figure A.26: $n = 20$, $d = 20$, Search: Time, $C > 0.5$, AC-3_d.

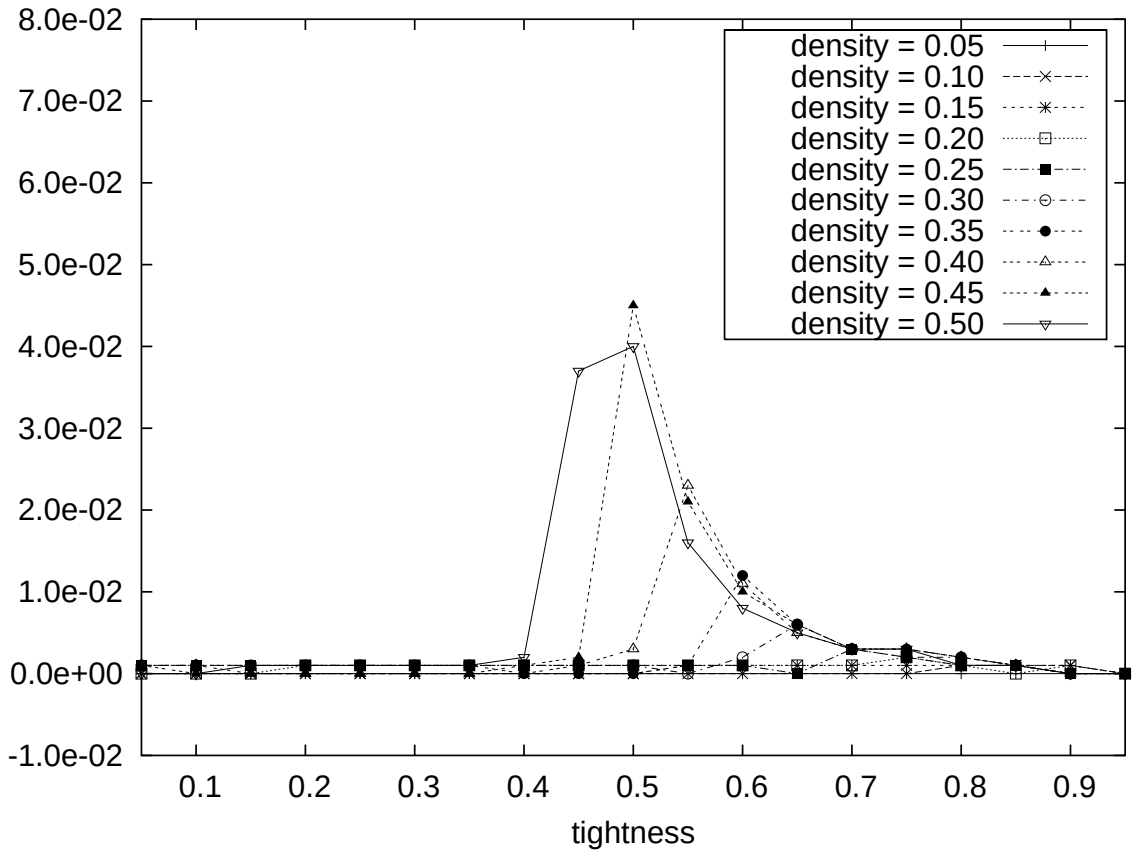

Figure A.27: $n = 20$, $d = 20$, Search: Time, $C \leq 0.5$, AC-2001 $-$ AC-3$_d$.

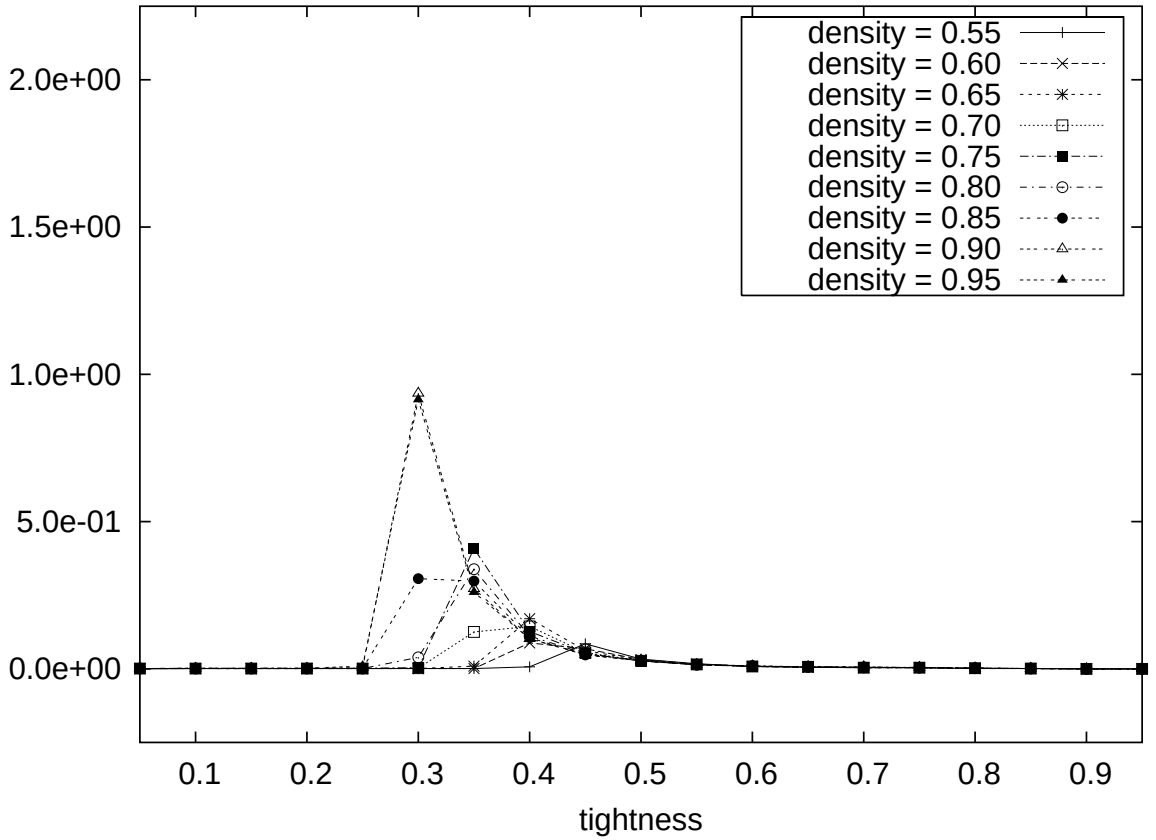

Figure A.28: $n = 20$, $d = 20$, Search: Time, $C > 0.5$, AC-2001 $-$ AC-3$_d$.

A.2 Results for $N = D = 30$

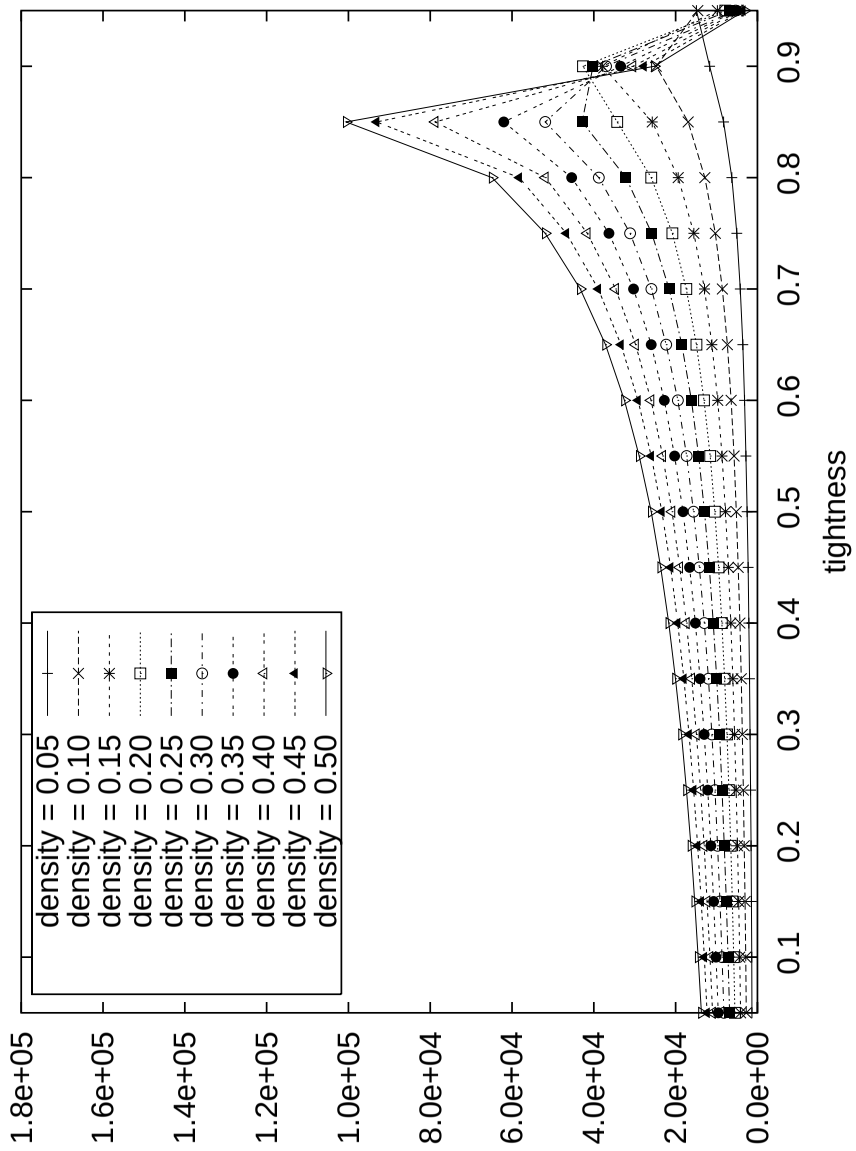

Figure A.29: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001.

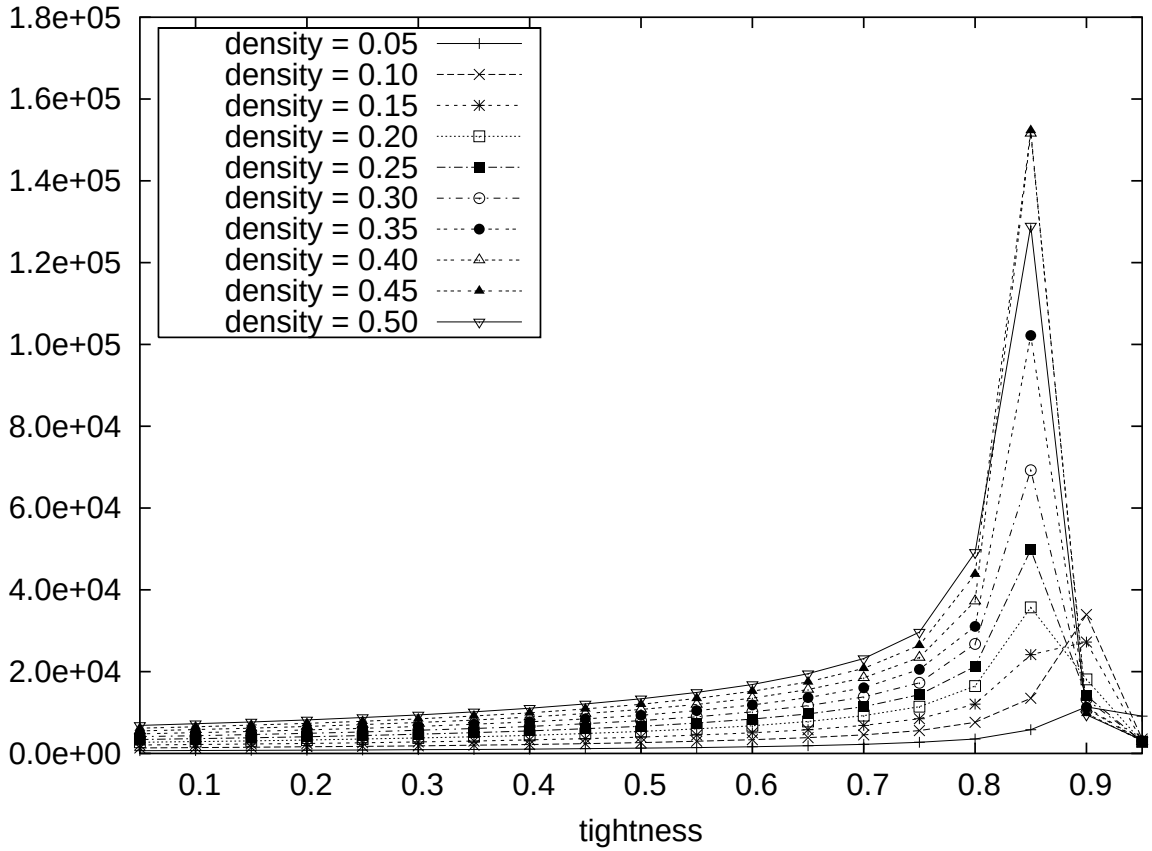

Figure A.30: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-3_d.

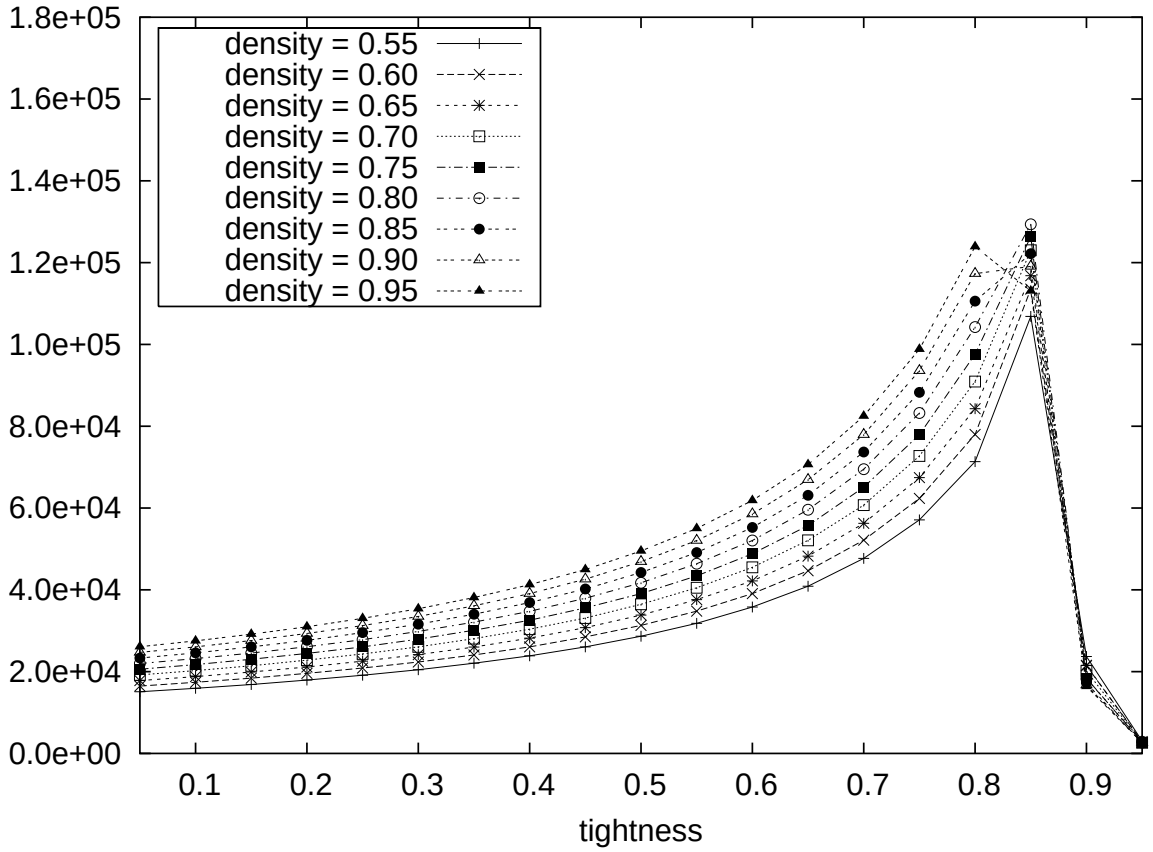

Figure A.31: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-2001.

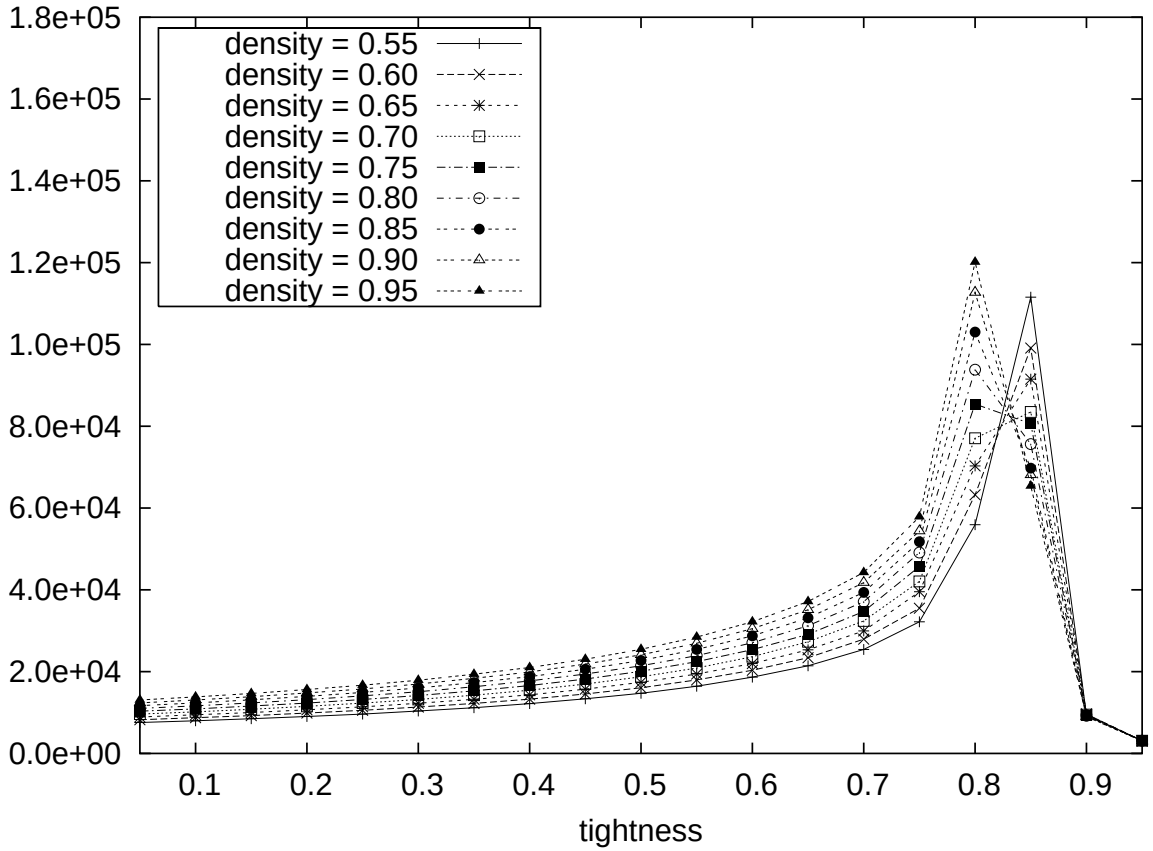

Figure A.32: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-3_d.

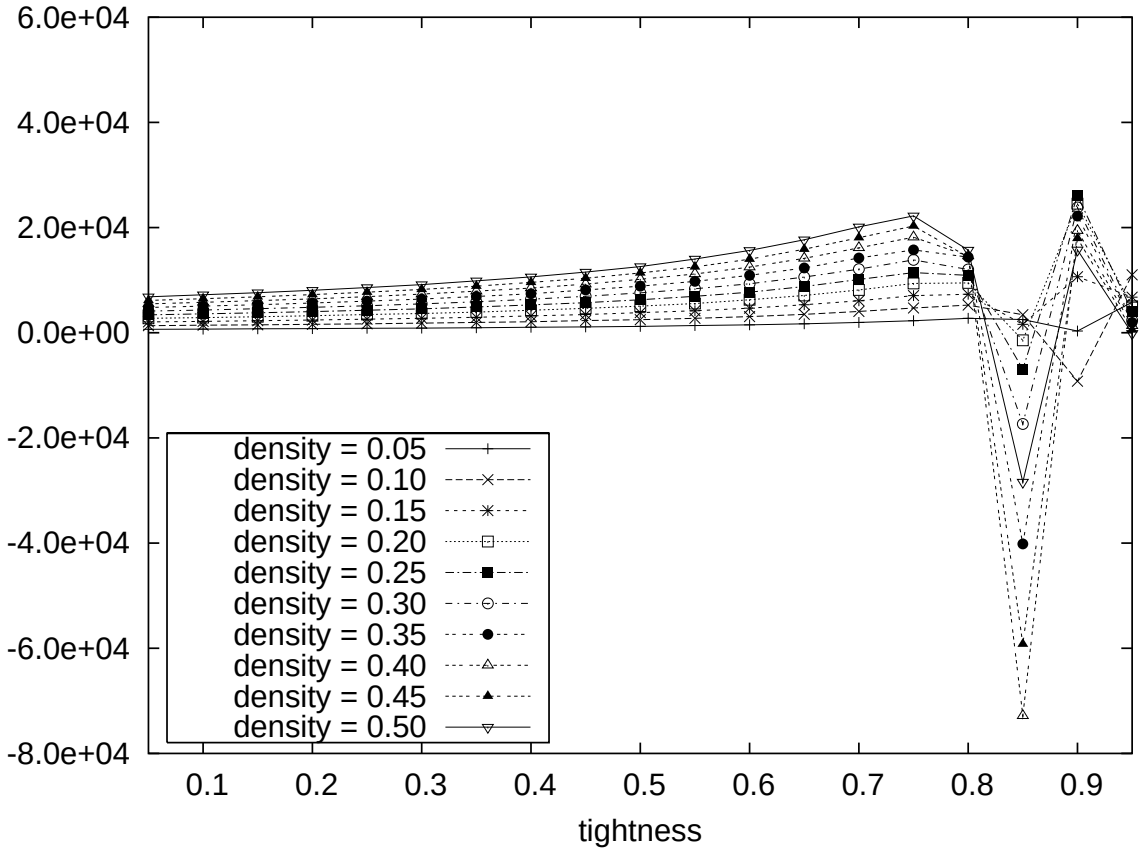

Figure A.33: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001 $-$ AC-3$_d$.

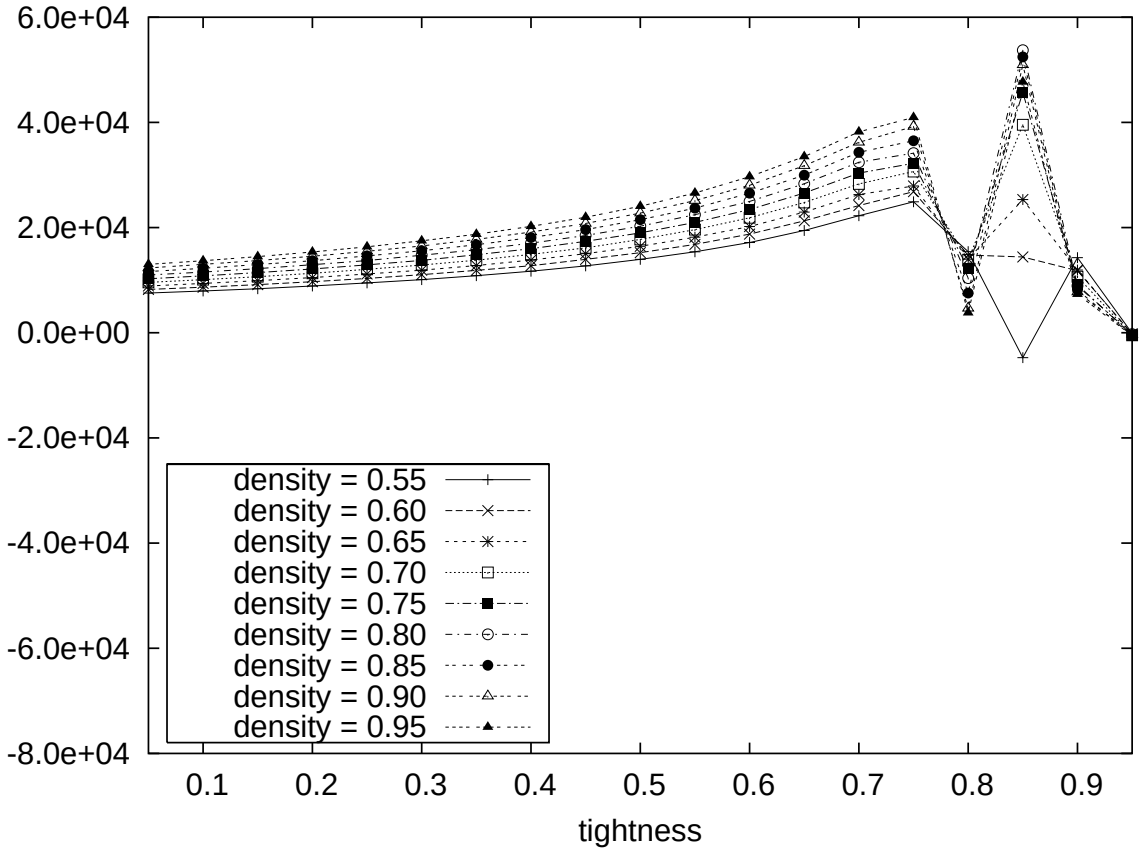

Figure A.34: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-2001 $-$ AC-3$_d$.

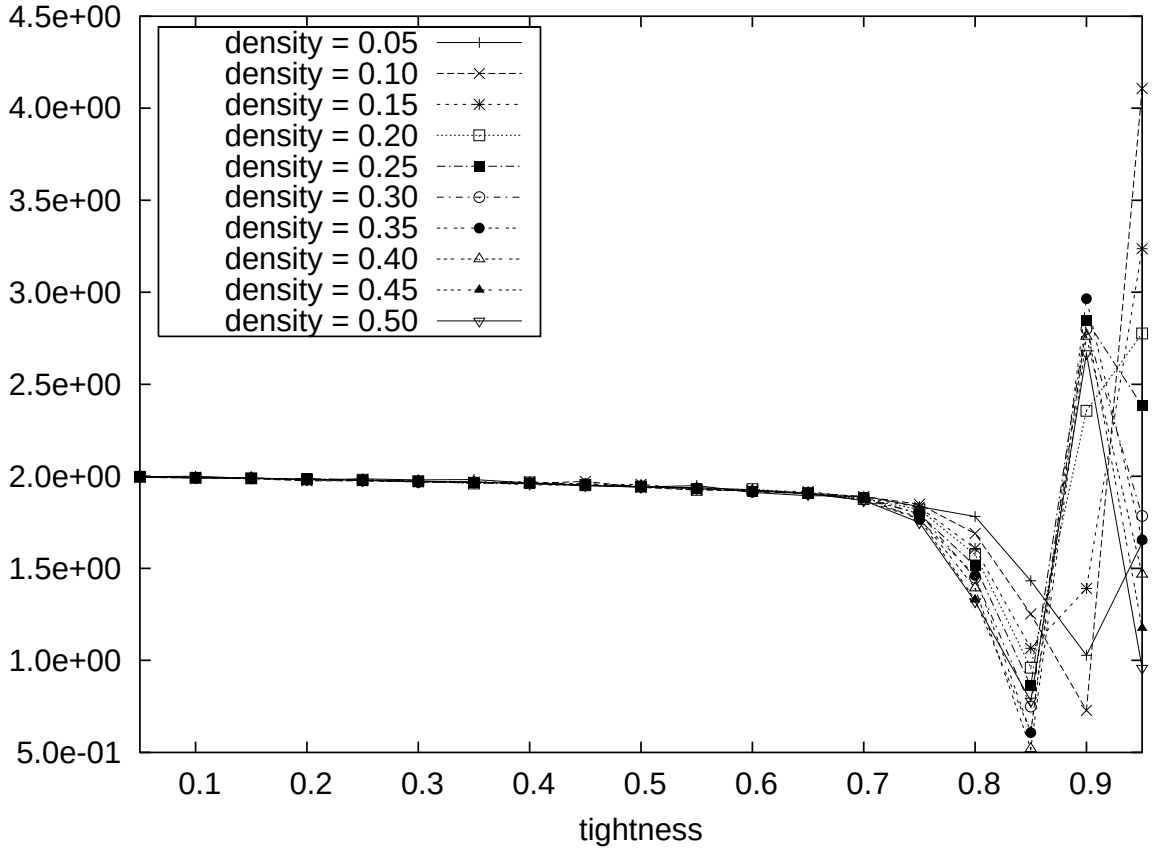

Figure A.35: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C \leq 0.5$, AC-2001/AC-3_d.

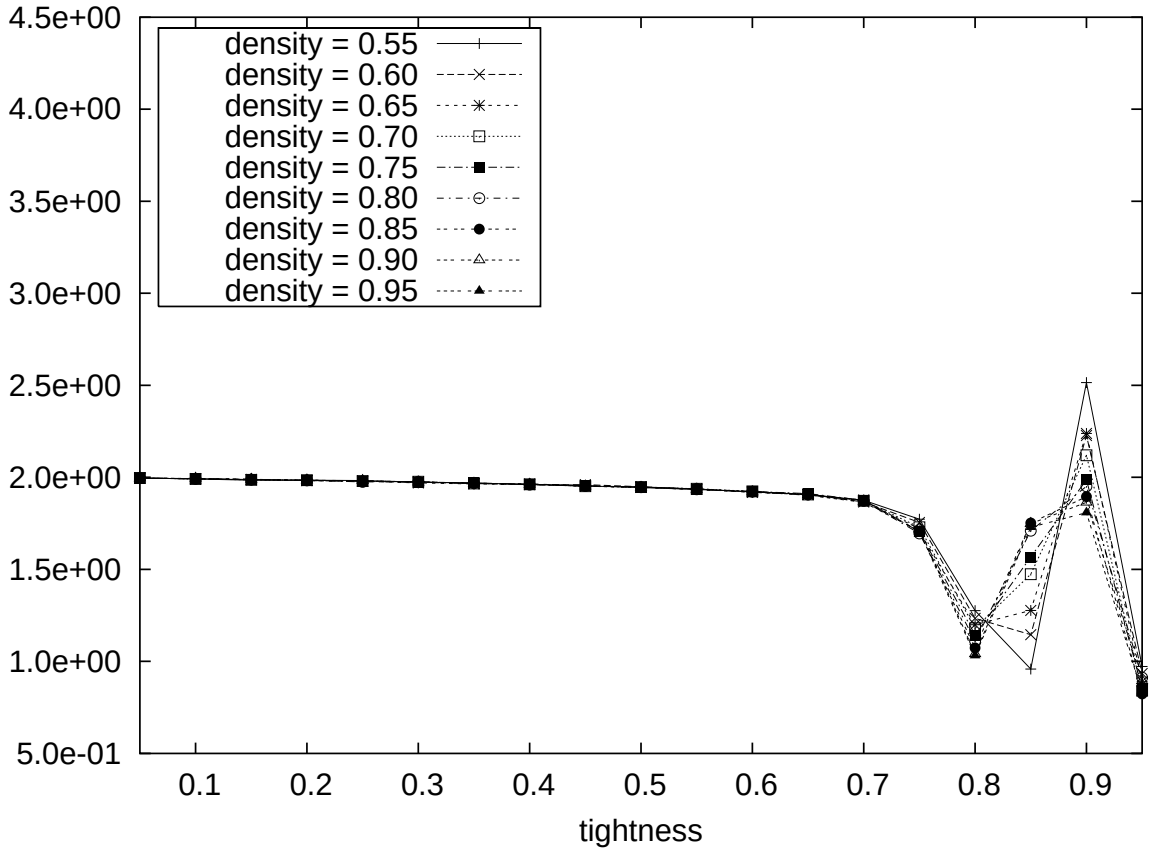

Figure A.36: $n = 30$, $d = 30$, Stand alone arc-consistency: Checks, $C > 0.5$, AC-2001/AC-3$_d$.

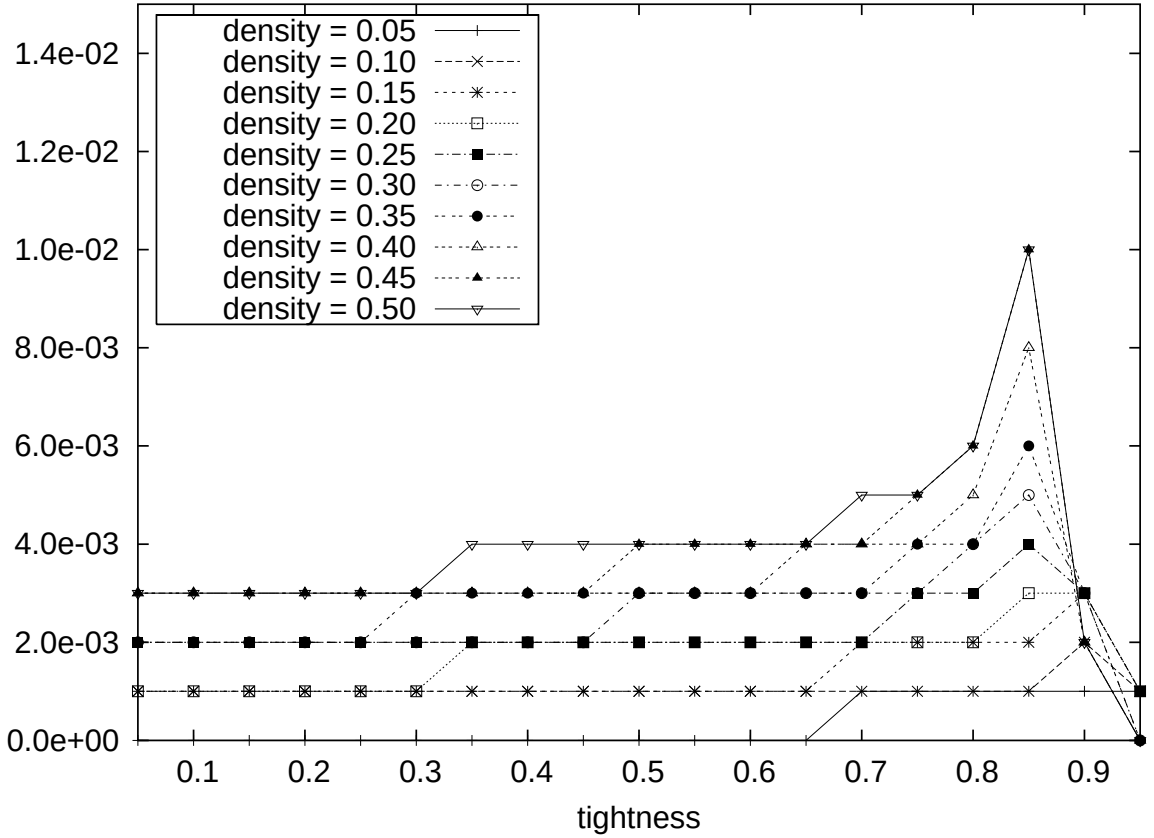

Figure A.37: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-2001.

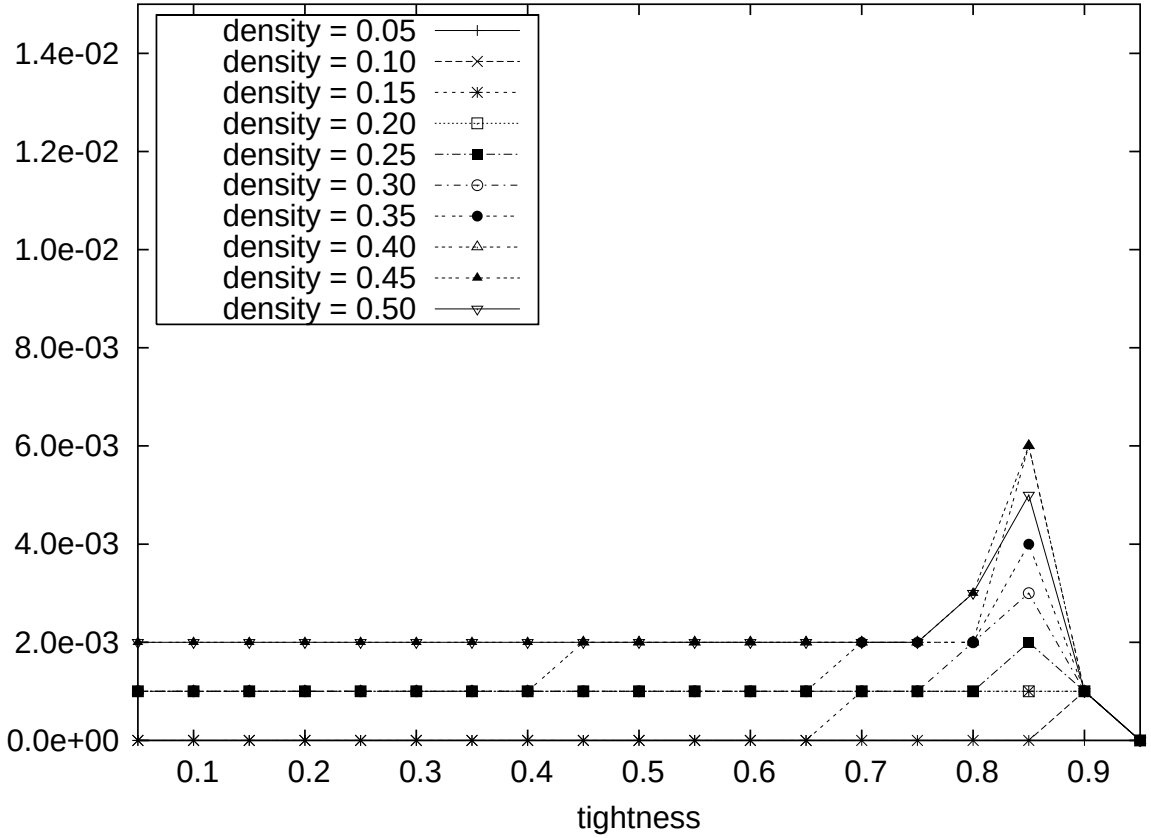

Figure A.38: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-3_d.

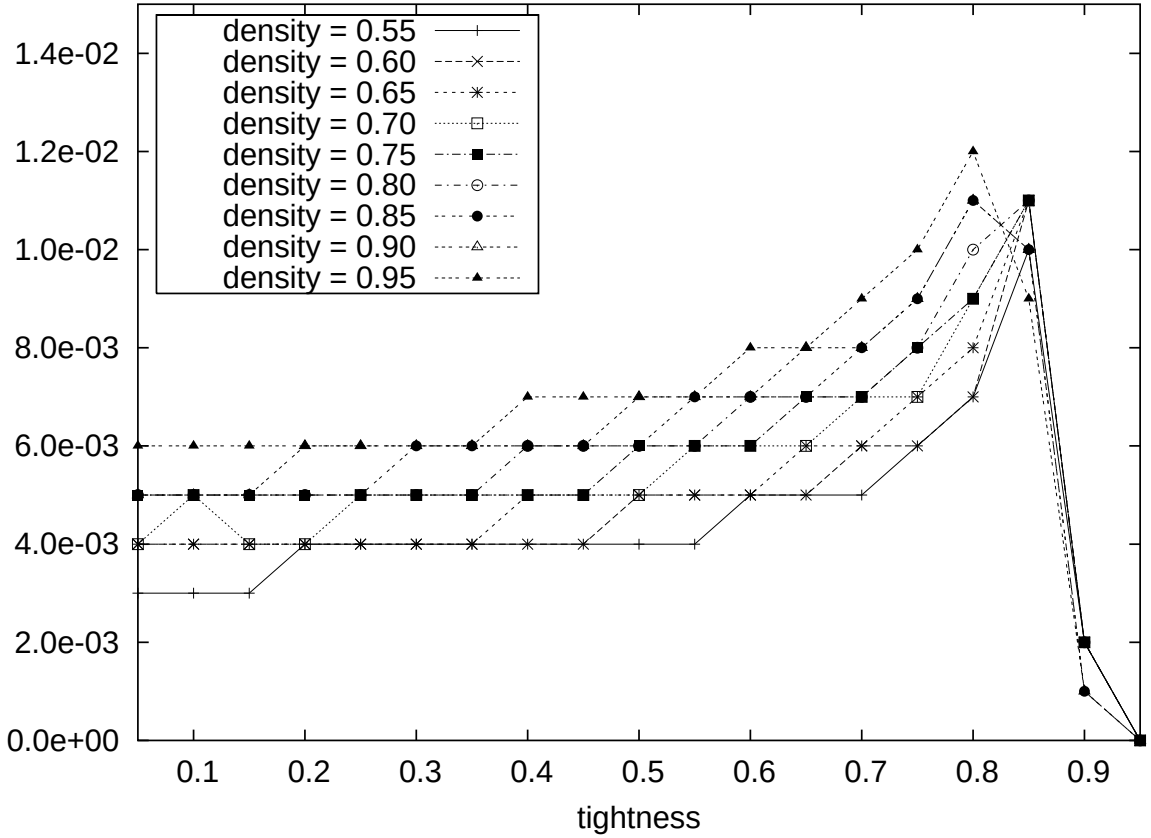

Figure A.39: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C > 0.5$, AC-2001.

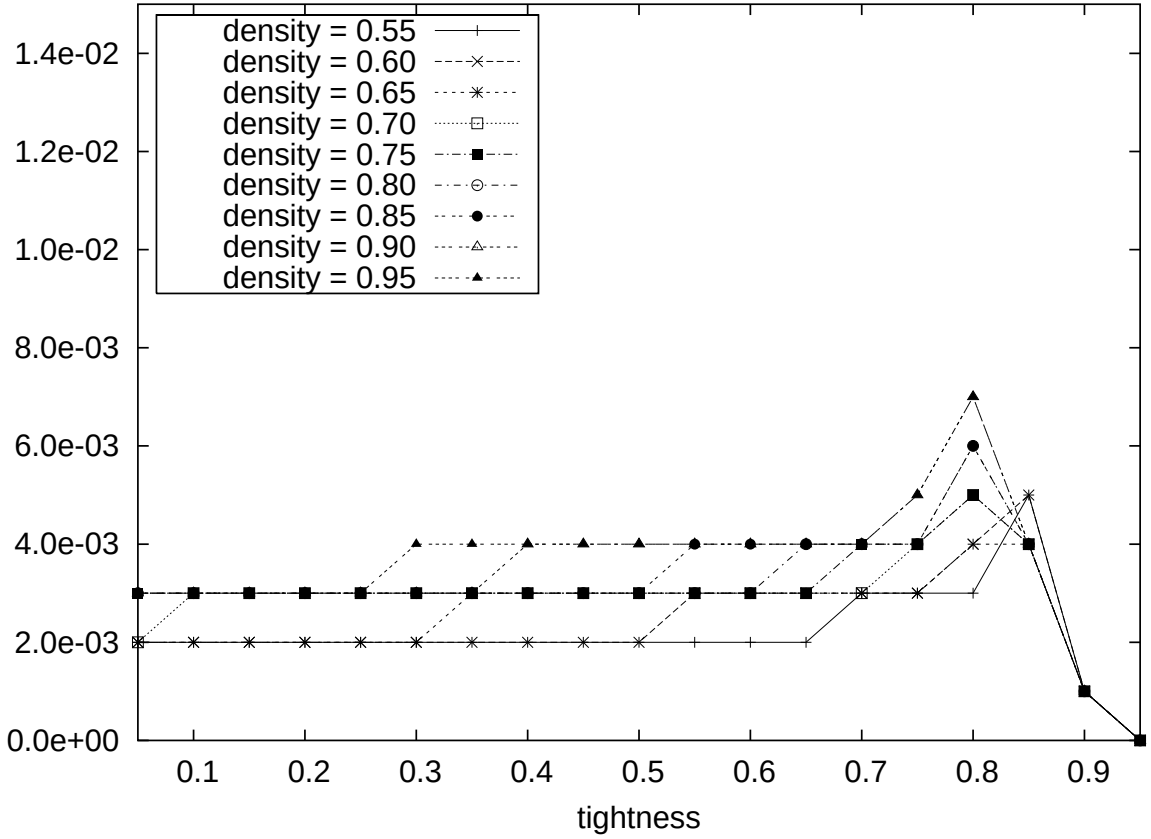

Figure A.40: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C > 0.5$, AC-3_d.

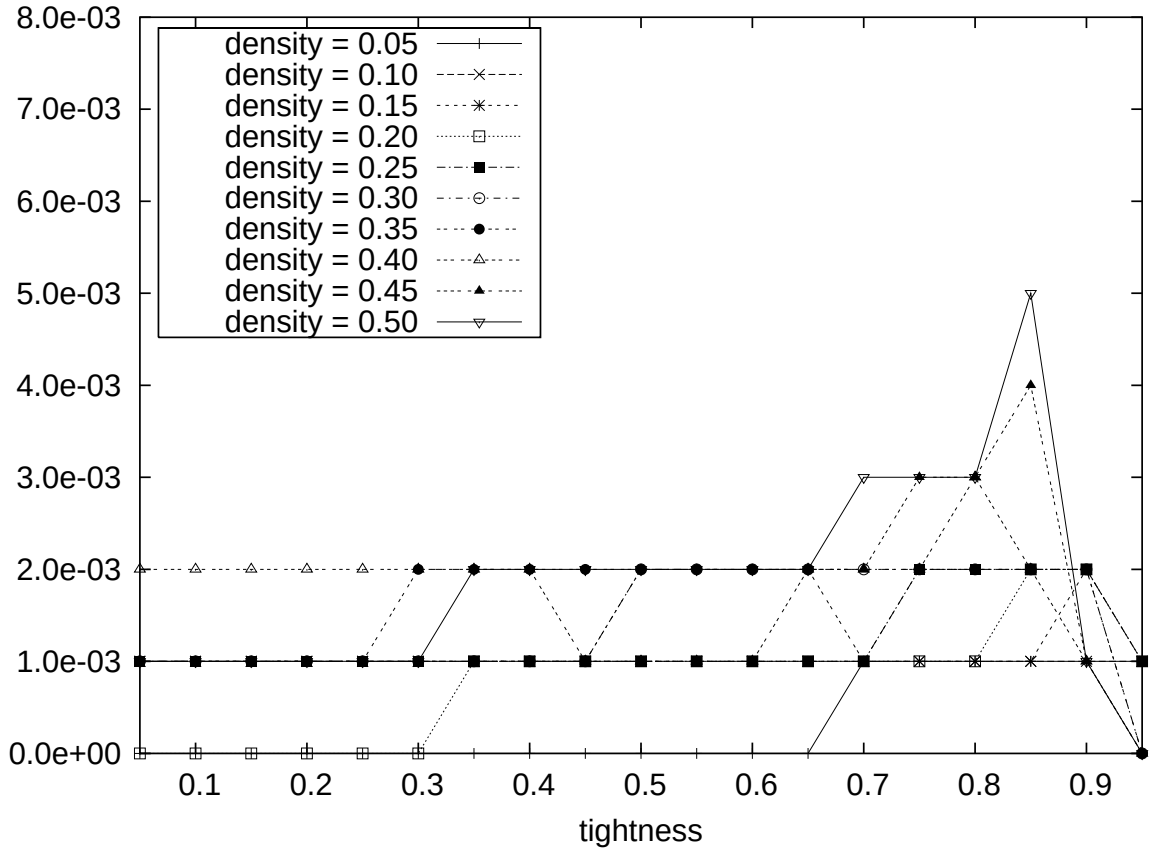

Figure A.41: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C \leq 0.5$, AC-2001 $-$ AC-3_d.

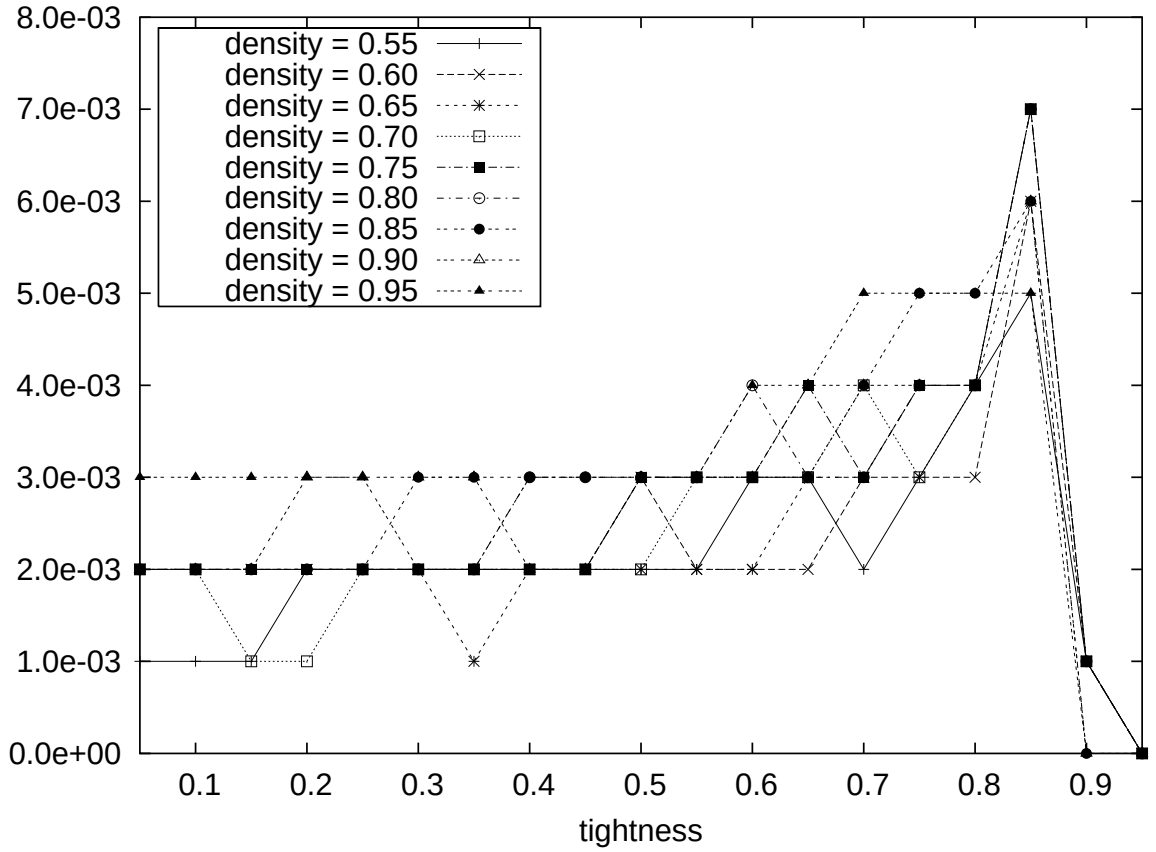

Figure A.42: $n = 30$, $d = 30$, Stand alone arc-consistency: Time, $C > 0.5$, AC-2001 – AC-3_d.

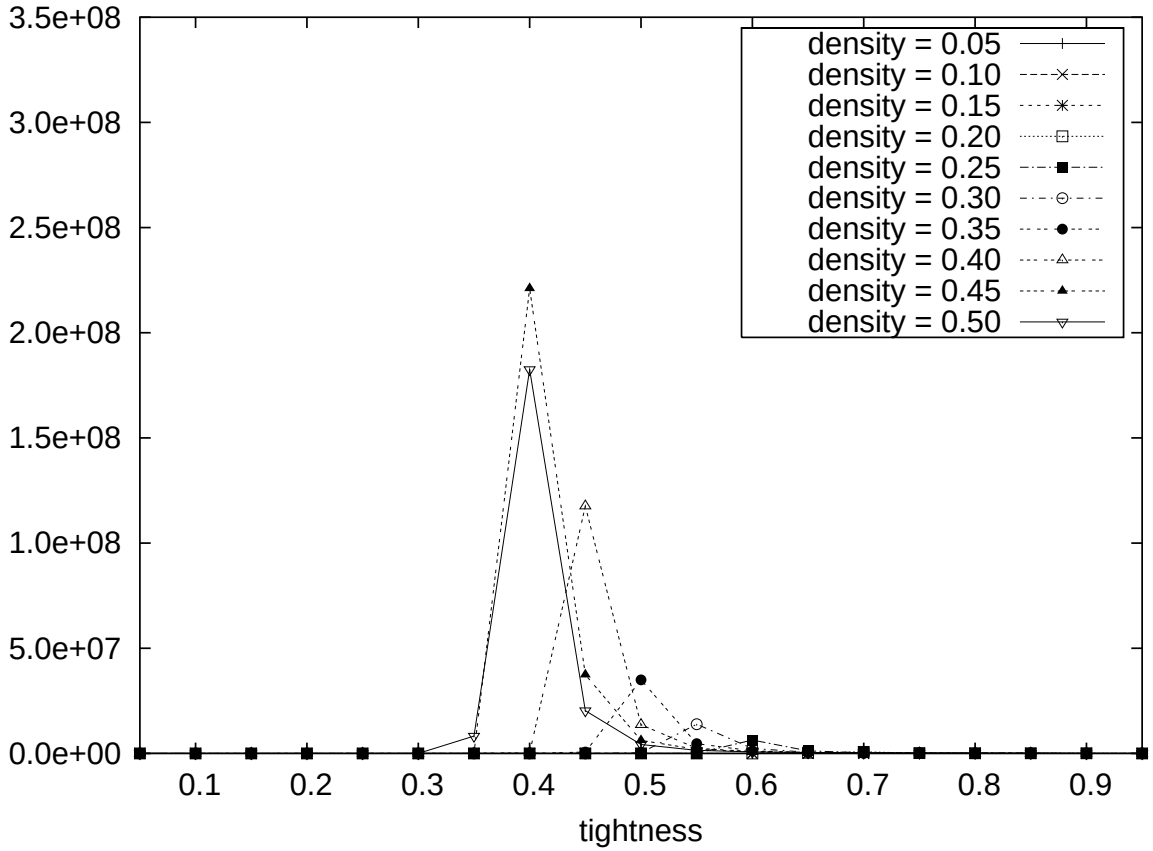

Figure A.43: $n = 30$, $d = 30$, Search: Checks, $C \leq 0.5$, AC-2001.

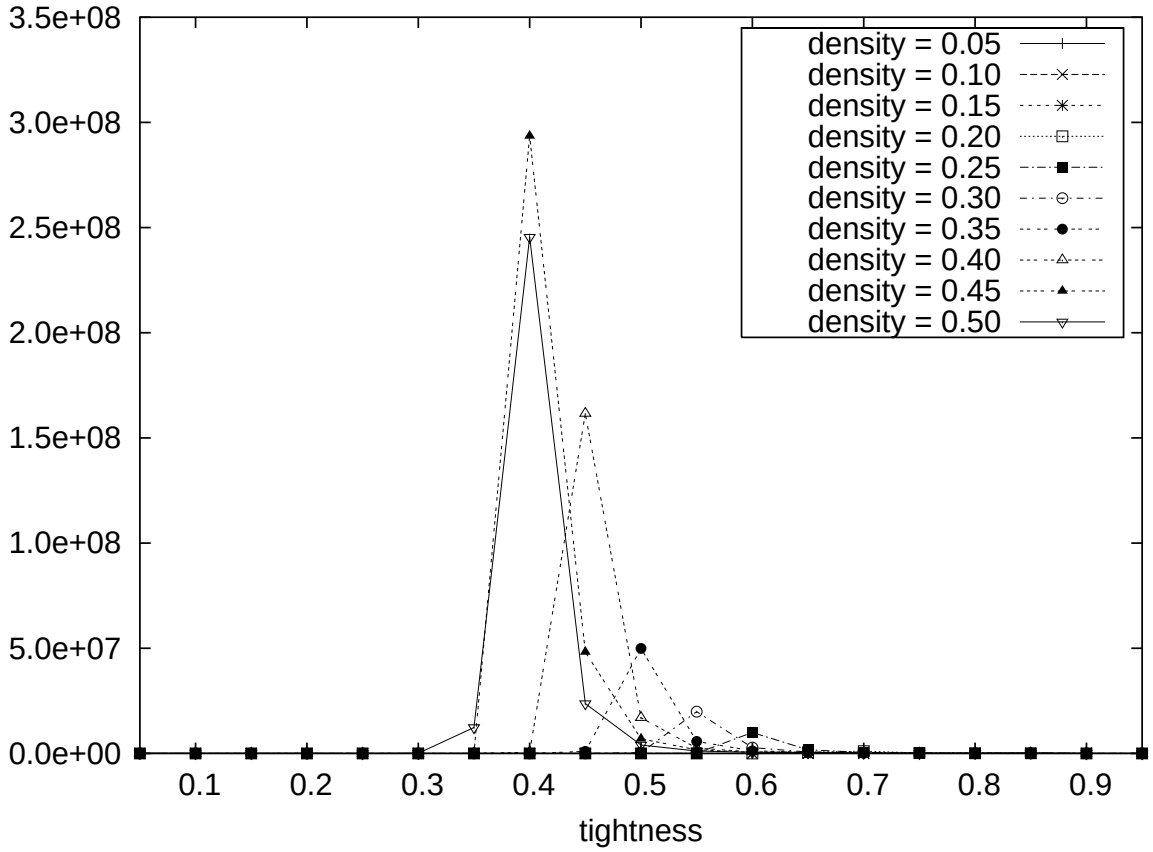

Figure A.44: $n = 30$, $d = 30$, Search: Checks, $C \leq 0.5$, AC-3_d.

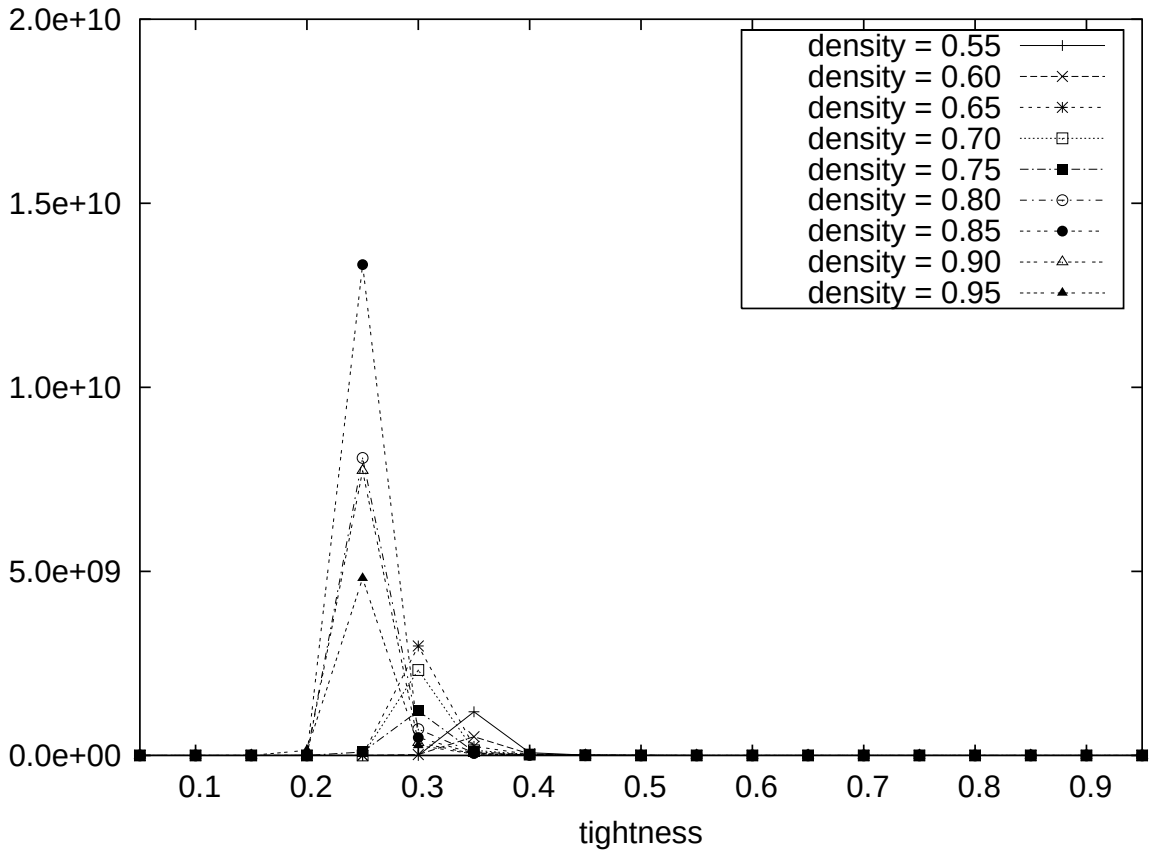

Figure A.45: $n = 30$, $d = 30$, Search: Checks, $C > 0.5$, AC-2001.

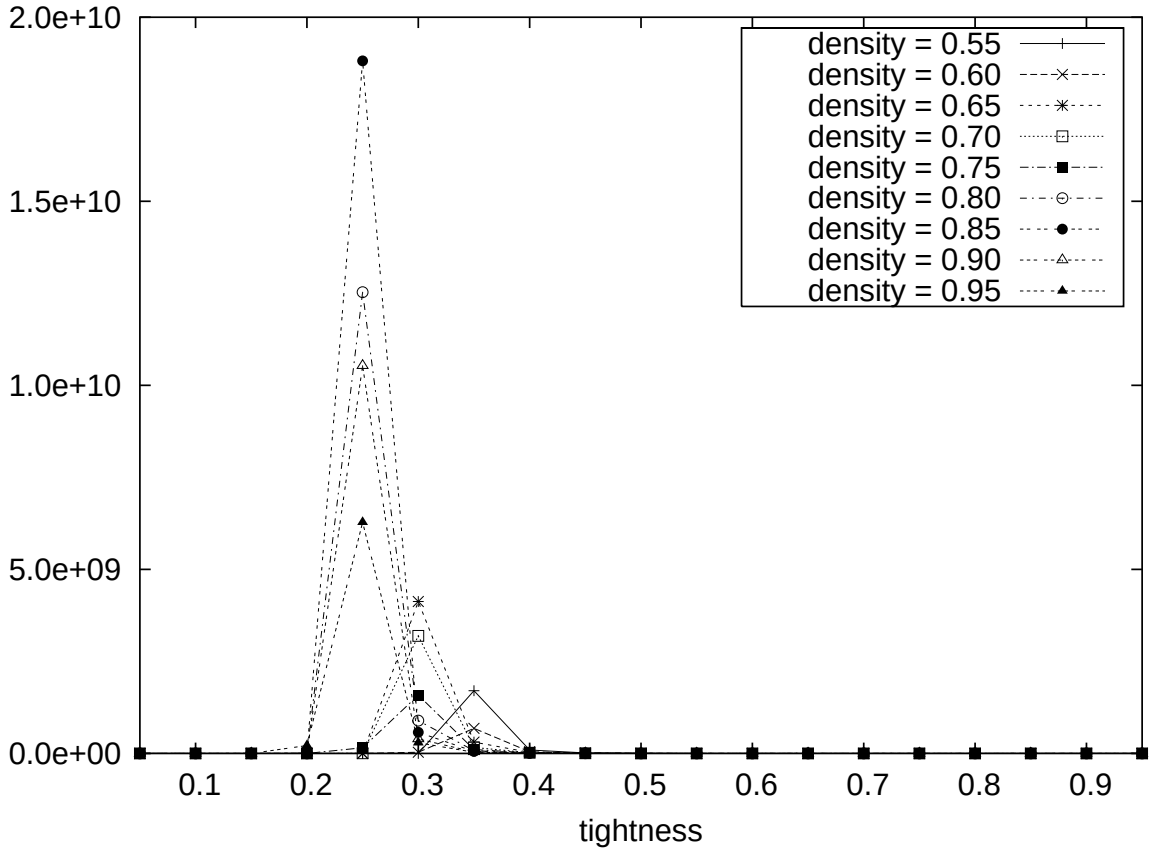

Figure A.46: $n = 30$, $d = 30$, Search: Checks, $C > 0.5$, AC-3_d.

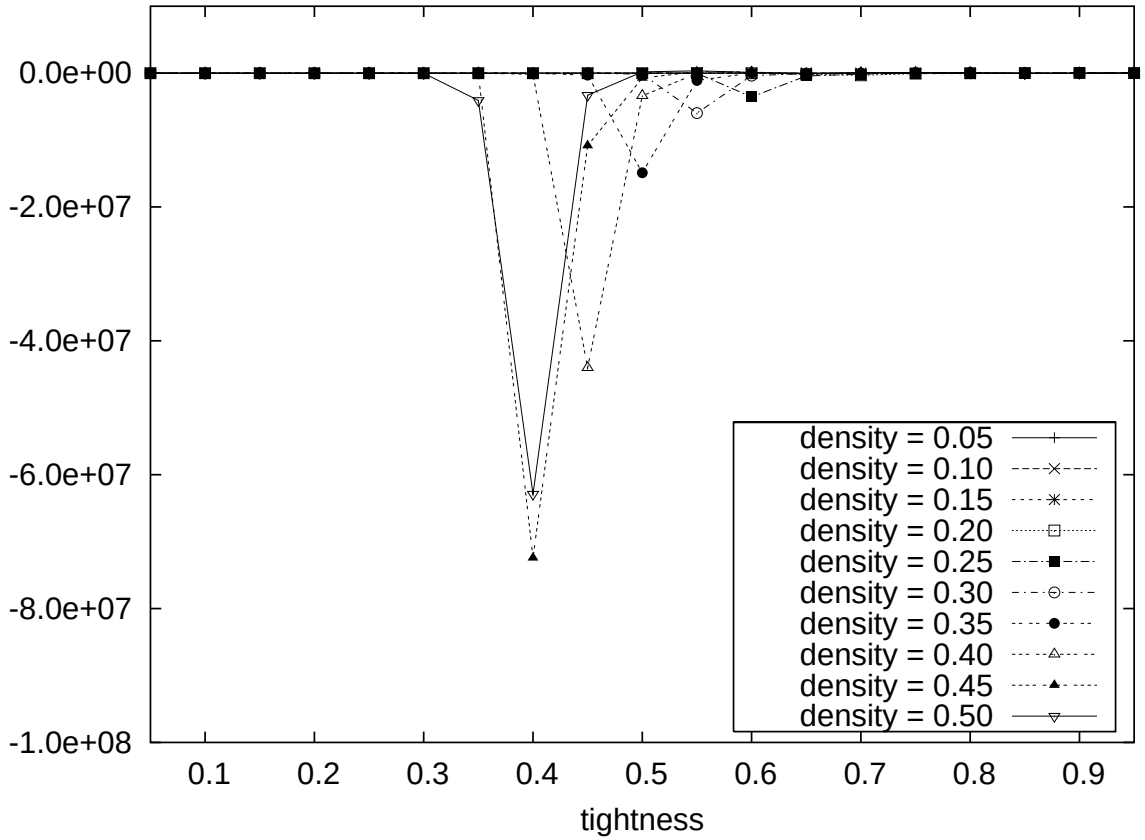

Figure A.47: $n = 30$, $d = 30$, Search: Checks, $C \leq 0.5$, AC-2001 $-$ AC-3$_d$.

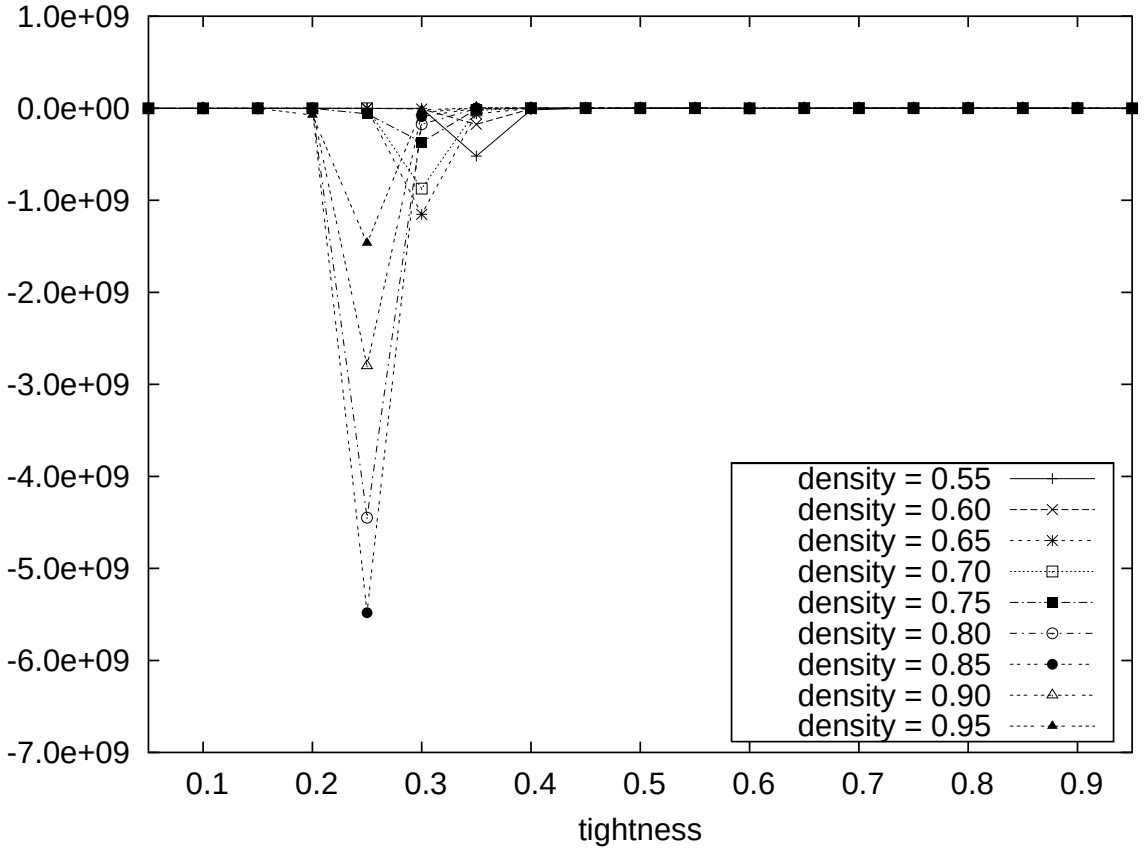

Figure A.48: $n = 30$, $d = 30$, Search: Checks, $C > 0.5$, AC-2001 $-$ AC-3$_d$.

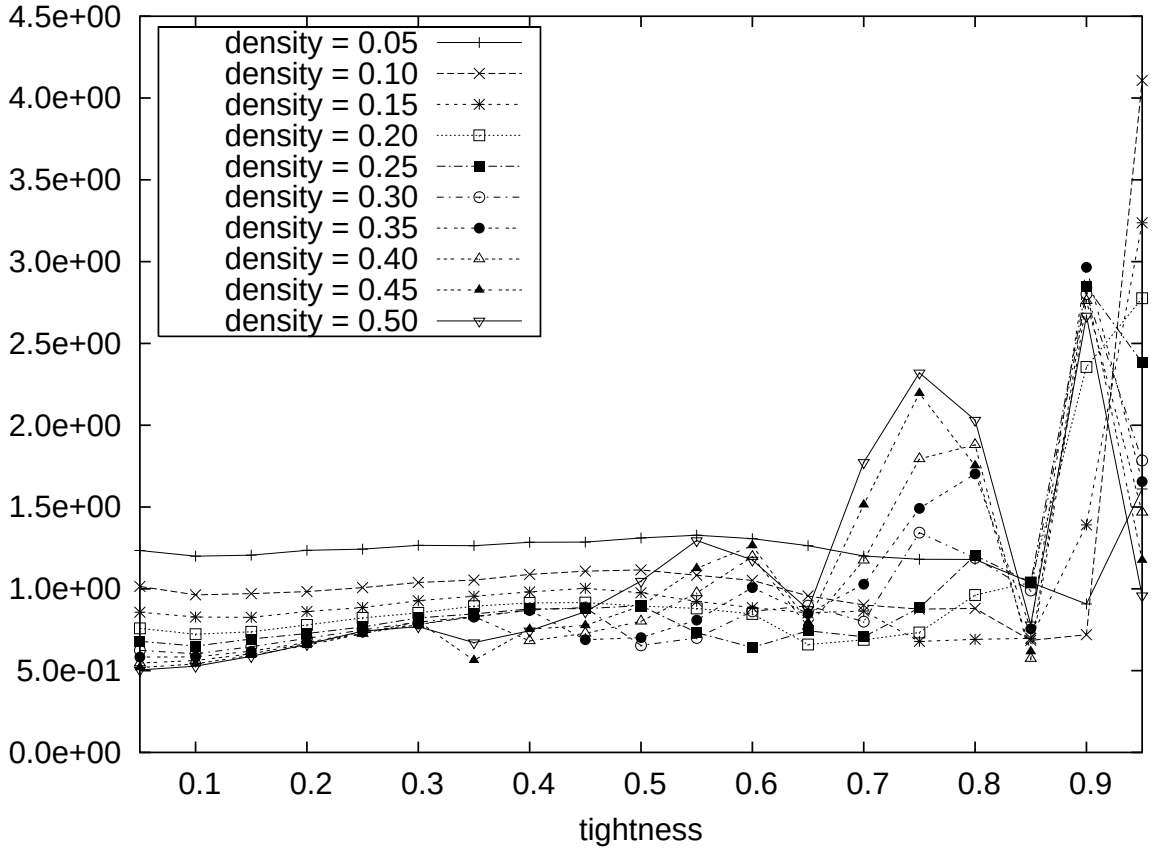

Figure A.49: $n = 30$, $d = 30$, Search: Checks, $C \leq 0.5$, AC-2001/AC-3$_d$.

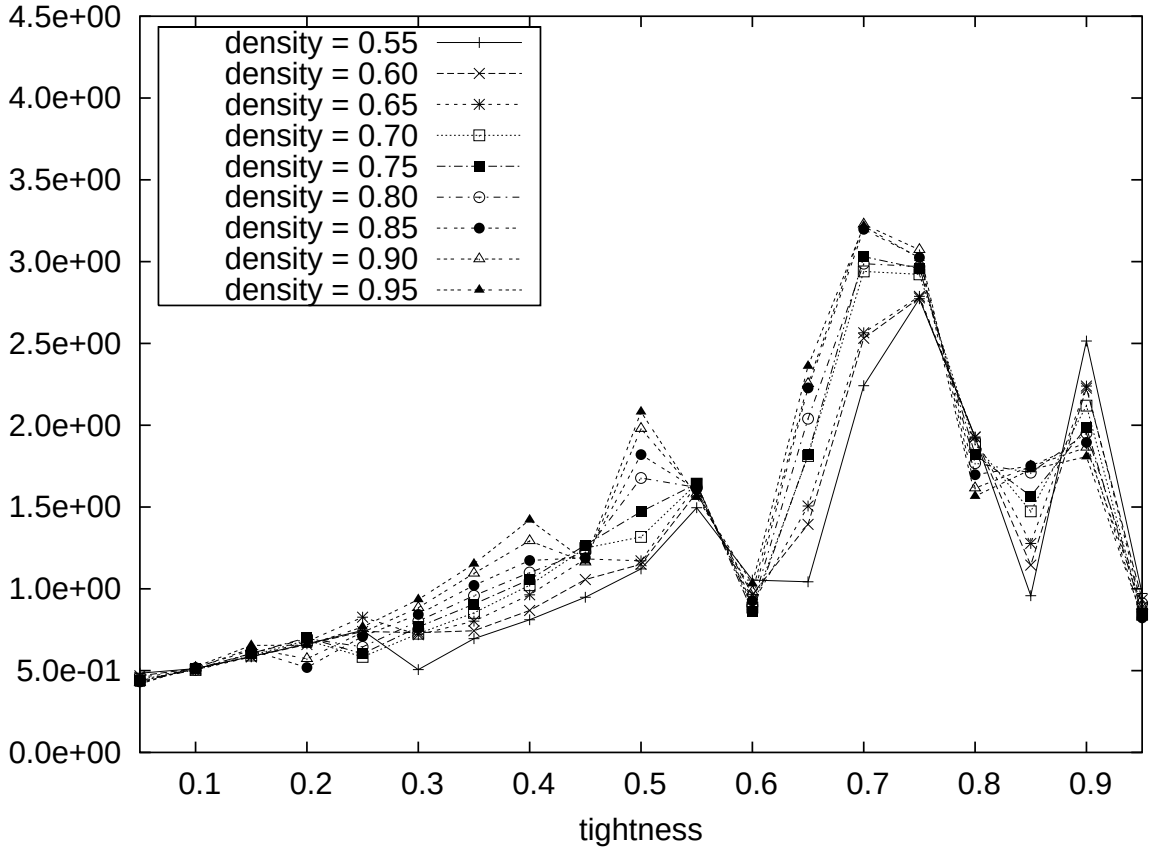

Figure A.50: $n = 30$, $d = 30$, Search: Checks, $C > 0.5$, AC-2001/AC-3$_d$.

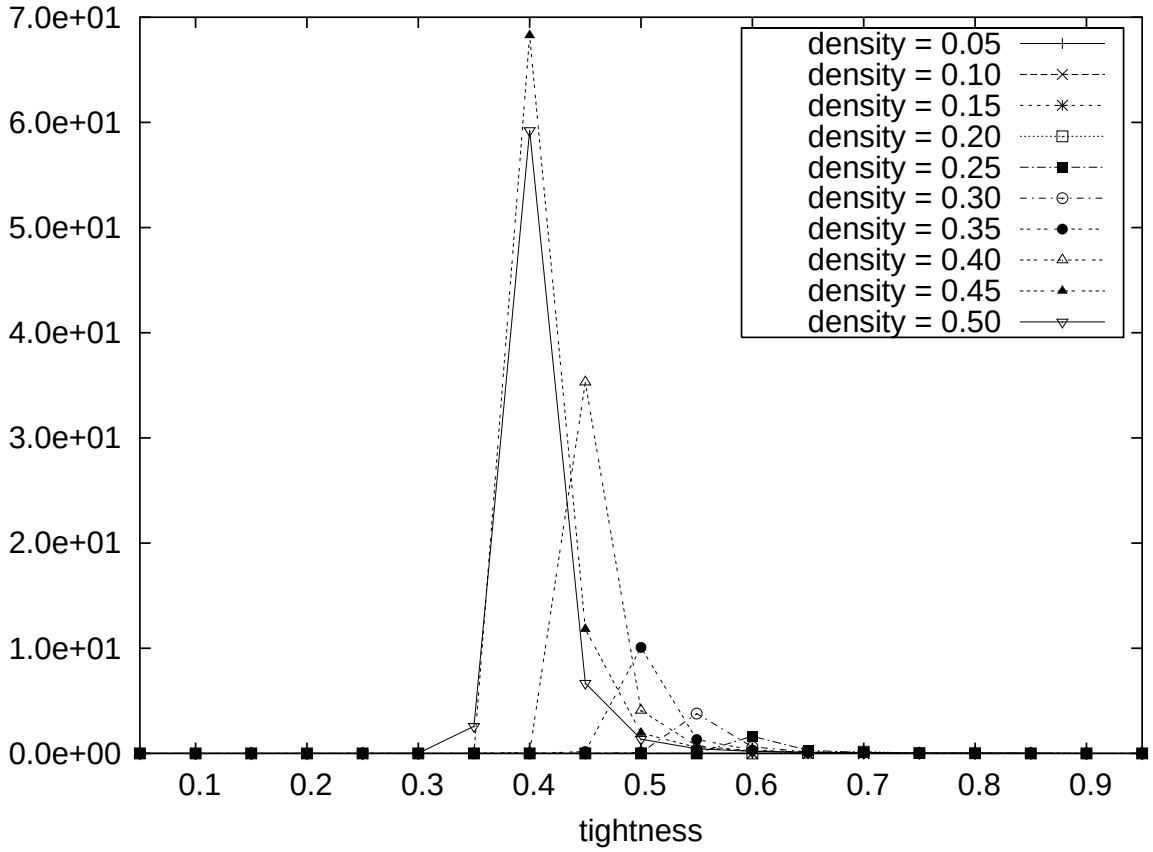

Figure A.51: $n = 30$, $d = 30$, Search: Time, $C \leq 0.5$, AC-2001.

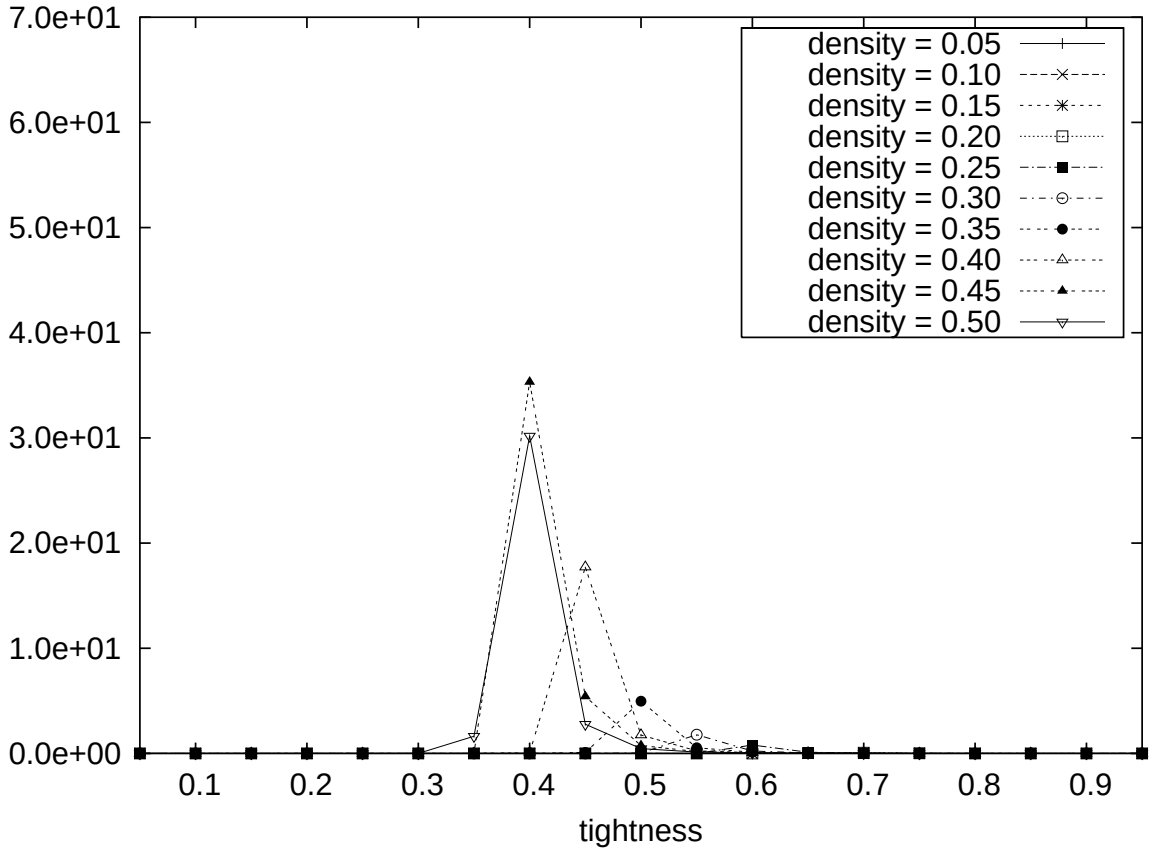

Figure A.52: $n = 30$, $d = 30$, Search: Time, $C \leq 0.5$, AC-3_d.

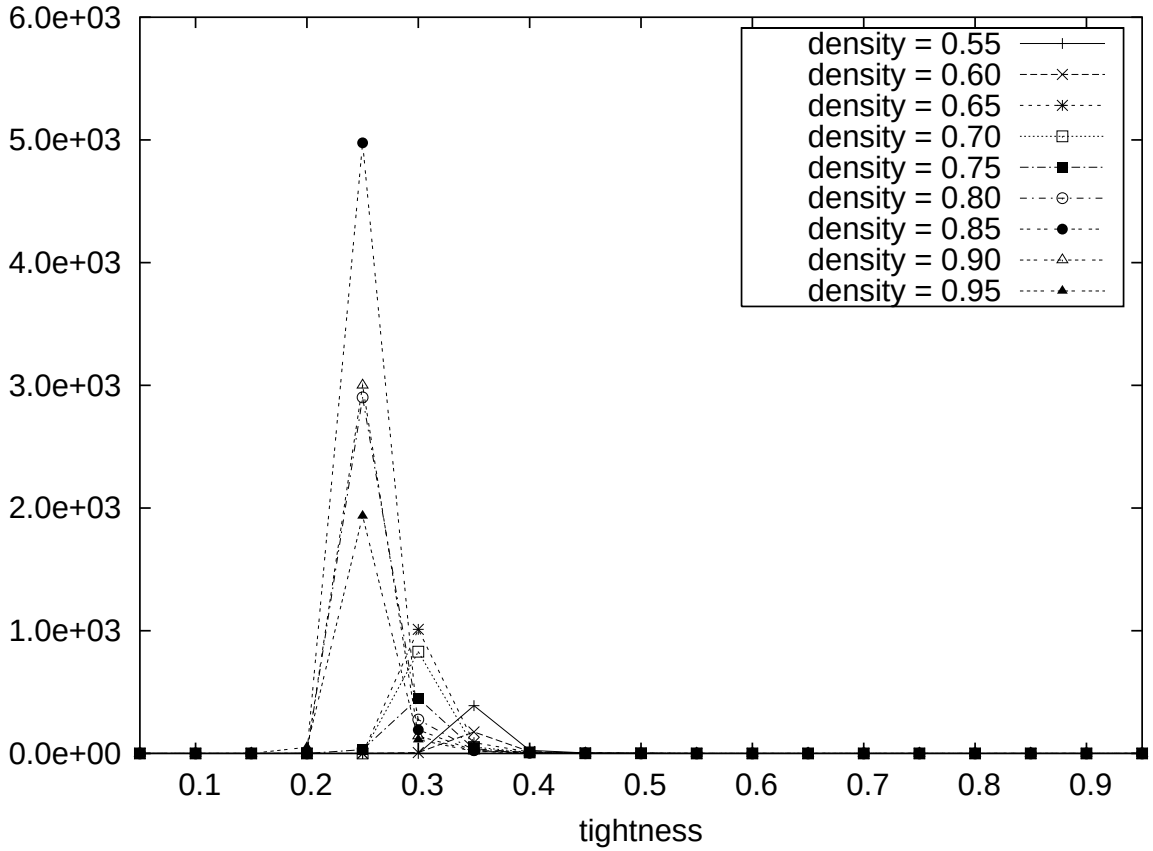

Figure A.53: $n = 30$, $d = 30$, Search: Time, $C > 0.5$, AC-2001.

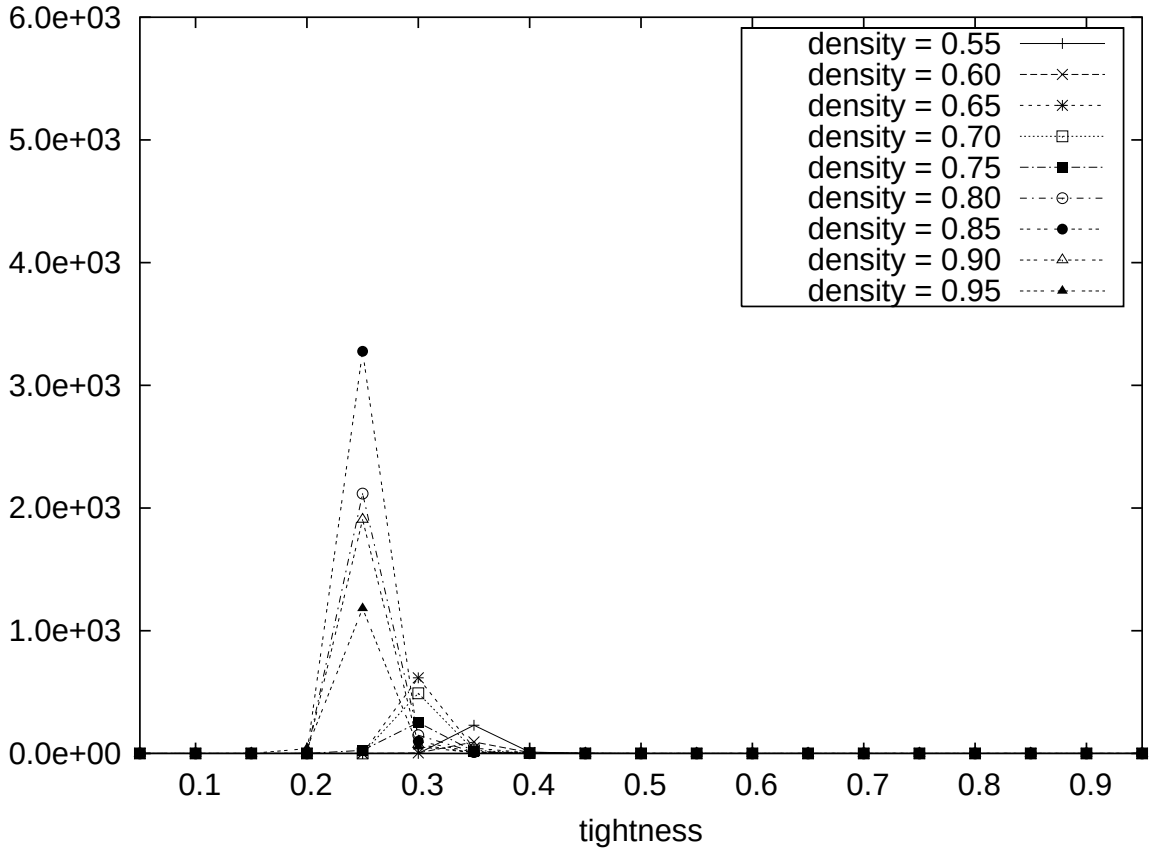

Figure A.54: $n = 30$, $d = 30$, Search: Time, $C > 0.5$, AC-3_d.

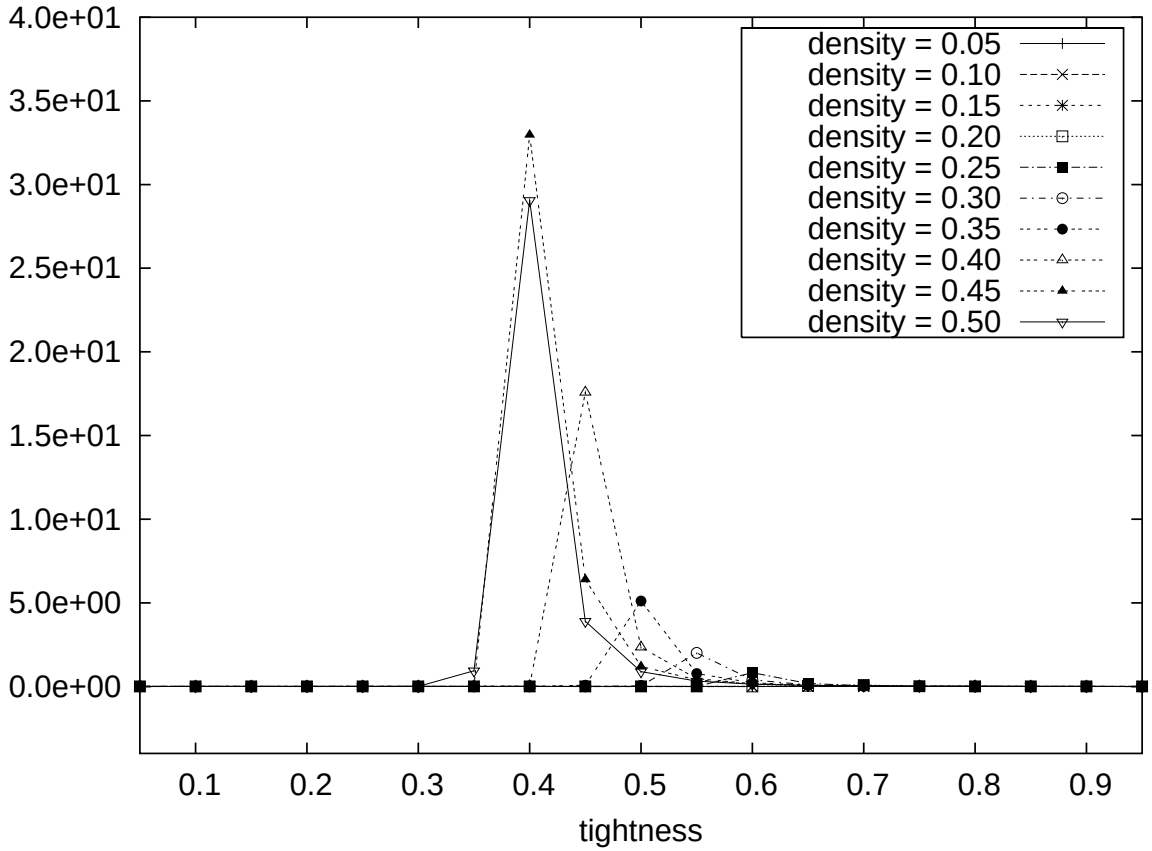

Figure A.55: $n = 30$, $d = 30$, Search: Time, $C \leq 0.5$, AC-2001 $-$ AC-3$_d$.

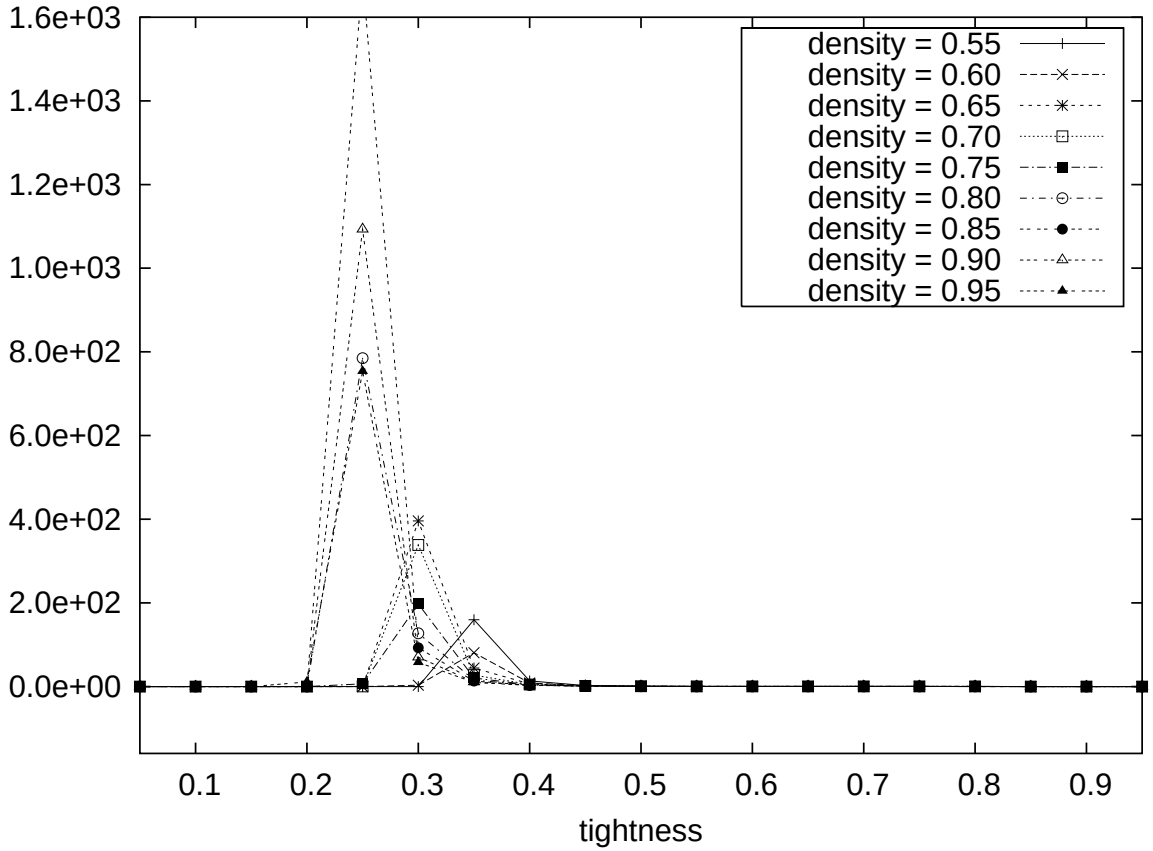

Figure A.56: $n = 30$, $d = 30$, Search: Time, $C > 0.5$, AC-2001 $-$ AC-3$_d$.