Negative Cycle Detection Problem

Chi-Him Wong and Yiu-Cheong Tam

The Chinese University of Hong Kong
{02658924, 02654543}@alumni.cse.cuhk.edu.hk

Abstract. In this paper, we will describe some heuristics that can be used to improve the runtime of a wide range of commonly used algorithms for the negative cycle detection problem significantly, such as Bellman-Ford-Tarjan (BFCT) algorithm, Goldberg-Radzik (GORC) algorithm and Bellman-Ford-Moore algorithm with Predecessor Array (BFCF). The heuristics are very easy to be implemented and only require modifications of several lines of code of the original algorithms. We observed that the modified algorithms outperformed the original ones, particularly in random graphs and no cycle graphs. We discovered that 69% of test cases have improved. Also, the improvements are sometimes dramatic, which have an improvement of a factor of 23, excluding the infinity case, while the worst case has only decreased by 85% only, which is comparably small when compared to the improvement.

1 The Negative Cycle Detection Problem

1.1 Introduction

The Negative Cycle Detection problem has numerous applications in model verification, compiler construction, software engineering, VLSI design, scheduling, circuit production, constraint programming and image processing. For example, Constraint-based program analysis requires feasibility checking of constraint sets. Constraint graphs are often used to represent systems of difference constraints; an application of Farkas' Lemma shows that a system of difference constraints is feasible if and only if there are no negative cost cycles in the corresponding constraint graph. That is, a difference constraint problem is feasible if and only if there are no negative cycles in the graph.

In the design of VLSI circuits, it is required to isolate negative feedback loops. These negative feedback loops correspond to negative cost cycles in the amplifier-gain graph of the circuit. The problem of checking whether a zero-clairvoyant scheduling system has a valid schedule can also be reduced to the problem of identifying negative cost cycles in the appropriate graph. Recent approaches to the image segmentation problem are also based on negative cycle detection. Most of the approaches are based on the famous Bellman-Ford (BF) algorithm. All

[0] The work described in this paper was partially supported by a direct allocation grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. 2050321).

G.S. Brodal and S. Leonardi (Eds.): ESA 2005, LNCS 3669, pp. 652–663, 2005.

these algorithms have their worst case bound of $O(|V||E|)$ in runtime. Most of the previous algorithms run relatively slow on no cycle graphs. However, no cycle graph is common in practice. So we try to develop some heuristics that are good in detecting no cycle graphs. Our heuristics can be used to improve the runtime of a wide range of Bellman-Ford based algorithms. We discovered that 69% of test cases have improved. Also, the improvements are sometimes dramatic, which have an improvement of a factor of 23, excluding the infinity case, while the worst case has only decreased by 85% only, which is comparably small when compared to the improvement. There are also significant improvement in no cycle graphs and random graphs.

2 Related Works

2.1 Definitions

The Negative Cycle Detection problem can be defined as the problem of deciding whether a negative cost cycle exists in a directed graph. This does not require finding a path from a particular source to a particular destination, as it is only a decision problem and require only a yes or no answer. Formally, the Negative Cycle Detection (NCD) problem is defined as follows:

Given a directed graph $G = < V, E, c >$*, where* $V = \{ v_0, v_1, \ldots, v_{n-1} \}$*,* $|V| = n$*,* $E = \{ e_{ij} : v_i \rightarrow v_j \}$*,* $|E| = m$*, and a cost function* $c : E \rightarrow Z$*, is there a negative cost cycle in* G *?*

There are no restrictions on the edge costs, i.e., they can be arbitrary integers as opposed to small integers, as required by some scaling algorithms. We can even extend the weights to floating point numbers.

Our heuristics can be applied to a wide range of commonly used algorithms for the Negative Cycle Detection problem that is a *single source algorithm*. In the following, we will define the meaning of single source algorithm.

An algorithm for the negative cycle detection problem is defined as a single source algorithm if and only if the algorithm is relaxation-based and all the relaxations (or calculations of labels) originate from a single vertex (the source).

We will give some examples of single source algorithms in the following.

2.2 The Bellman-Ford Moore Algorithm with Predecessor Array (BFCF)

The Bellman-Ford-Moore (BFFI) algorithm attempts to reduce the number of vertices that must be examined in each stage of the “standard” Bellman-Ford (BF) algorithm by using a First-In-First-Out (FIFO) queue to store the vertices whose distance labels were changed in the previous stage.

Predecessor Array is a strategy that uses parent pointer to store the parent of each vertex v_i, where the parent of v_i is the vertex that caused the most recent label change to v_i. This strategy is widely used in many Bellman-Ford based algorithms.

The Bellman-Ford-Moore algorithm with Predecessor Array (BFCF) is an algorithm that combines the above two techniques. As the algorithm starts with a vertex with label 0 and all the other vertices of label infinity, it is a single source algorithm. Notice that BFCF still has the worst case bound of $O(|V||E|)$ in runtime.

2.3 The Bellman-Ford-Tarjan Algorithm (BFCT)

Another variation of the BF algorithm is BFCT, which combines the "standard" BF algorithm with the FIFO queue of BFFI and the sub-tree disassembly cycle detection strategy due to Tarjan. The sub-tree disassembly strategy is implemented by describing the tree structure by, besides the usual predecessor function, the first son and the adjacent brother function (next and previous). This allows traversals of sub-trees in linear time, and tree modifications in constant time. When no cycle is found in the traversal of a sub-tree T_v, all vertices in T_v will be discarded from the queue and the tree. [Tar81]. A negative cycle will be detected when a tree path from a vertex goes back to one of its ancestors. As the algorithm is started with one source of label 0 and all the other vertices of labels infinity, it is a single source algorithm.

2.4 The Goldberg-Radzik Algorithm (GORC)

The Goldberg-Radzik algorithm improved the Bellman-Ford-Moore algorithm. It achieves the same worst-case bound of $O(|V||E|)$, but can usually outperform BFFI in practice. The algorithm maintains the set of labelled vertices in two queues, queue A and queue B. Vertices in queue A will undergo a Bellman-Ford-Moore pass while vertices in queue B will undergo a depth first search with no update. At the beginning of the algorithm, the source is put in queue B. In pass B, if there is an outgoing arc with reduced cost including zero, the path is traversed and all vertices visited will be put in queue A. Notice that there are no updates on the vertices' labels in this step. Also, if a reduced path from a vertex goes back to one of its ancestors, a negative cycle is detected. On the other hand, pass A is a one step Bellman-Ford-Moore pass that relaxes the vertices in queue A following the queue order. Notice that topological sort will be done after the depth first searches in pass B, so we will relax the vertices in pass A topologically and the label will be updated.

The algorithm starts with one source of label 0 and all the other vertices of labels infinity, it is a single source algorithm.

3 Our New Approach to This Problem

Our approach is to incorporate two heuristics to the above single source algorithms.

3.1 Heuristic One: Pumping Negative Strategy

The first heuristic is based on an observation that in any negative cost cycle, we can find a negative weighted edge e such that when we travel through the negative cost cycle starting from e, the accumulated costs are always negative. The formal definition of this lemma is as follows.

Lemma 1: Given a directed graph $G = < V, E, c >$, *where* $V = \{ v_0, v_1, \ldots, v_{n-1} \}$, $|V| = n$, $E = \{ e_{ij} : v_i \rightarrow v_j \}$, $|E| = m$, *and c is cost function* $E \rightarrow Z$, *with a negative cost cycle* $\{ v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)}, \ldots, v_{\pi(k)} \}$, *where* $v_{\pi(1)} = v_{\pi(k)}$, *there exists at least one node* $v_{\pi(i)}$ *where* $1 \leqq i \leqq k$ *such that when we travel through the negative cost cycle starting from* $v_{\pi(i)}$, *the accumulated costs are always negative.*

In this heuristic, we will stop the relaxation once the label becomes positive. But since we do not know which vertex in the negative cycle is the starting vertex with the property as stated in Lemma 1, we need to treat each vertex as source once. The proof of Lemma 1 is shown in Appendix II.

3.2 Heuristic Two: Reduced Cost Elimination

This heuristic is simply not deleting the labels on the vertices when moving from one pass to another with different vertices as the source. Consider a case where a particular vertex relaxes another vertex with label that is not marked 0. If the label is smaller than the accumulated cost, we will have the following two conclusions:

1. That particular vertex has been travelled previously as it has a label that is not 0.
2. The accumulated cost of the previous travel must be smaller than current accumulated cost as the label is smaller than the current accumulated cost.

Therefore, as the current accumulated cost is larger, using the current accumulated cost will reduce the number of vertices visited. Therefore, we can keep the cost label.

4 Modifications on Various Algorithms

In this section, we will briefly describe how we implement our heuristics into some single source algorithms.

4.1 Modification of BFCF

For heuristic one, we first change the algorithm such that the original BFCF algorithm will run V times with a different vertex as the source each time. We also add a condition for the relaxation process which is, a relaxation can only be done and continued if the accumulated cost is negative. When there are no more relaxations, a new source will be chosen and the whole process is repeated. For heuristic two, we can implement it simply by keeping the cost label on each vertex unchanged when we start a new pass with a new source. The pseudo-code is as follows:

Initialize all cost labels to 0. /*So, all accumulated cost will be negative (Heuristic 1)*/

```
For all v /*(Heuristic 1)*/
{
BFCF with v as the source /*(Heuristic 1)*/
/*Keep all the labels unchanged (Heuristic 2)*/
}
```

4.2 Modifications of BFCT

The modifications to BFCT are similar to that in BFCF. However, each vertex in BFCT will be involved in the first son and adjacent brother functions (next and previous). During our modification, we can leave the values of the function (next and previous) unchanged when we choose a new starting point. This will improve the runtime without affecting the correctness.

4.3 Modifications of GORC

The modifications for GORC are a bit different from that of BFCF and BFCT. For heuristic one, we need to modify the condition in pass B to ensure that the depth first search traversal will only traverse an edge when it leads to a negative reduced cost. GORC has already implemented the reduced cost elimination heuristic, so we do not need to do any modification for heuristic two.

5 Experimental Setups

5.1 Results

The problem generator was developed by the authors of [Gol95]. It can be downloaded from the website http://www.avglab.com/andrew/index.html. There are several sets of data.

1. The Rand-5 families have a fixed network size n=2000000 and m=10000000. The maximum arc length U is fixed at 32000 and the minimum arc length L varies from 0 to –64000 [Gol95].
2. The SQNC families represent the square grid families. Vertices of these networks correspond to points on the $x - y$ plane with integer coordinates [x, y], $0 \leqq x \leqq X$, $0 \leqq y \leqq Y$ and $X = Y$. The SQNC01 family has no cycles. The SQNC02 family has sparse small negative cycles. The SQNC03 family has dense small negative cycles. The SQNC04 family has several long cycles. The SQNC05 family has Hamilton cycle.
3. The LNC families are the grid networks mentioned above but with Y=16. The LNC01, LNC02, LNC03, LNC04 and LNC05 are similar to those in the SQNC families.
4. The PNC families are the layered networks. A layered network consists of layers 0… X-1. Each layer is a simple cycle plus a collection of arcs connecting randomly selected pairs of vertices on the cycle. In the PNC families each layer contains 32 vertices and $X = n/32$ [Gol95]. The PNC01, PNC02, PNC03, PNC04 and PNC05 are similar to those in the SQNC families.

There are 5 test cases for each row shown in the following tables. The number of scan operations shown is the average of the 5 test cases. The percentage improvement is calculated according to the number of operation of relaxing. Results with more than 10% faster or slower will be highlighted in different colors. Finally, the 'M' in front of the name of the algorithm implies the modified algorithm with our heuristics implemented. The results are shown in Appendix I.

6 Conclusion

In this paper, we described two heuristics that can be incorporated into a wide range of commonly used single source algorithms for the Negative Cycle Detection problem. The modifications are very simple, involving only adding several lines of code.

After modification, all algorithms have increased the speed generally. There are significant results in the set of random graphs and no cycle graphs. We discovered that 69% of test cases have improved. Also, the improvements are sometimes dramatic, which have an improvement of a factor of 23, excluding the infinity case, while the worst case has only decreased by 85% only, which is comparably small when compared to the improvement.

References

[CG96] Boris V. Cherkassky and Andrew V. Goldberg. Negative-cycle detection algorithms. In Josep D'ıaz and Maria Serna, editors, Algorithms—ESA '96, Fourth Annual European Symposium, volume 1136 of Lecture Notes in Computer Science, pages 349–363, Barcelona, Spain, 25–27 September 1996. Springer.

[Gol95] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. SIAM Journal on Computing, 24(3):494–504, June 1995.

[Tar81] R. E. Tarjan. Shortest Paths. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1981.

Appendix I

Table 1. Results for no cycle graphs LNC01

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
LNC01	8193	29571	51894	75.5%	14891	27338	83.6%
	16385	59399	102501	72.6%	29864	54115	81.2%
	32769	117508	204668	74.2%	59016	107378	81.9%
	65537	236006	412421	74.8%	118558	217019	83.0%
	131073	471187	824874	75.1%	236706	433352	83.1%
	262145	940478	1643797	74.8%	472542	864805	83.0%
	524289	1881286	3289308	74.8%	945099	1733480	83.4%
	1048577	3764090	6590393	75.1%	1890963	3455877	82.8%
	2097153	7523534	13186716	75.3%	3779505	6917921	83.0%
	4194305	15052684	26342784	75.0%	7561213	13826436	82.9%
	8388609	30111238	52697650	75.0%	15127398	27660704	82.9%
	16777217	60193979	58728907	-2.4%	30237773	30997195	2.5%
Average				68.32%			76.11%

Table 2. Results for no cycle graphs PNC01

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
PNC01	8193	0	141440	INF	0	101024	INF
	16385	0	282502	INF	0	206452	INF
	32769	0	577308	INF	0	421004	INF
	65537	0	1175657	INF	0	840400	INF
	131073	0	2326792	INF	0	1697588	INF
	262145	0	4662634	INF	0	3378583	INF
	524289	0	9360087	INF	0	6776352	INF
	1048577	0	18799475	INF	0	13545258	INF
	2097153	0	37359269	INF	0	27079159	INF
	4194305	0	74824647	INF	0	54191807	INF
	8388609	0	149660040	INF	0	108381976	INF
Average				INF			INF

Table 3. Results for no cycle graphs SQNC01

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
SQNC01	4097	13503	29080	115.4%	6933	12259	76.8%
	16385	51932	114828	121.1%	26775	48554	81.3%
	65537	203158	479398	136.0%	104971	191840	82.8%
	262145	807586	1961943	142.9%	418223	733359	75.4%
	1048577	3211258	7930611	147.0%	1663635	2919808	75.5%
	4194305	12817258	32338613	152.3%	6643025	11873024	78.7%
	16777217	51213266	140101492	173.6%	26550354	48035413	80.9%
Average				141.19%			78.77%

Table 4. Results for a few short cycle graphs LNC02

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
LNC02	8193	6442	11480	78.2%	3240	5825	79.8%
	16385	19433	34692	78.5%	9716	17916	84.4%
	32769	42381	46765	10.3%	21241	24144	13.7%
	65537	90512	114548	26.6%	45277	61044	34.8%
	131073	151996	267506	76.0%	75979	141937	86.8%
	262145	247744	438869	77.1%	123912	231623	86.9%
	524289	696032	1163851	67.2%	348365	616440	77.0%
	1048577	1563907	1669857	6.8%	782080	883473	13.0%
	2097153	1845286	3252535	76.3%	923584	1729789	87.3%
	4194305	6011875	6697666	11.4%	3007849	3560722	18.4%
	8388609	13820476	24466542	77.0%	6911521	12969000	87.6%
	16777217	22339356	37514660	67.9%	11172724	19902492	78.1%
Average				54.44%			62.32%

Table 5. Results for a few short cycle graphs PNC02

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
PNC02	8193	47185	59183	25.4%	29484	30705	4.1%
	16385	86450	64450	-25.4%	53720	54659	1.7%
	32769	225235	205400	-8.8%	140052	142698	1.9%
	65537	287805	203174	-29.4%	178108	180636	1.4%
	131073	350055	294782	-15.8%	218465	212419	-2.8%
	262145	503878	386424	-23.3%	314458	330199	5.0%
	524289	1316216	1095546	-16.8%	820598	874808	6.6%
	1048577	4790290	3915894	-18.3%	2983431	3175931	6.5%
	2097153	10428337	8593237	-17.6%	6495202	6953796	7.1%
	4194305	23124100	19013193	-17.8%	14408767	15390086	6.8%
	8388609	67316841	55413705	-17.7%	41913396	44775974	6.8%
Average				-15.05%			4.10%

Table 6. Results for a few short cycle graphs SQNC02

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
SQNC02	4097	13503	29080	115.4%	6933	12259	76.8%
	16385	51932	114828	121.1%	26775	48554	81.3%
	65537	203158	479398	136.0%	104971	191840	82.8%
	262145	807586	1961943	142.9%	418223	733359	75.4%
	1048577	3211258	7930611	147.0%	1663635	2919808	75.5%
	4194305	12817258	32338613	152.3%	6643025	11873024	78.7%
	16777217	51213266	140101492	173.6%	26550354	48035413	80.9%
Average				141.19%			78.77%

Table 7. Results for many short cycles graphs LNC03

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
LNC03	8193	343	396	15.3%	173	196	13.2%
	16385	71	136	93.2%	31	47	48.4%
	32769	112	275	145.9%	54	72	33.1%
	65537	208	262	26.2%	106	97	-8.5%
	131073	180	221	22.3%	92	91	-0.7%
	262145	138	297	115.5%	72	82	14.2%
	524289	240	349	45.4%	120	110	-8.0%
	1048577	198	224	13.4%	97	69	-28.7%
	2097153	149	313	110.3%	74	73	-1.1%
	4194305	172	344	99.7%	86	86	-0.2%
	8388609	216	261	20.9%	106	102	-3.6%
	16777217	193	202	4.6%	95	100	5.7%
Average				59.39%			5.32%

Table 8. Results for many short cycles graphs PNC03

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
PNC03	8193	461	2713	489.1%	327	865	164.6%
	16385	473	3046	543.5%	256	451	75.7%
	32769	468	3843	721.2%	248	513	106.9%
	65537	341	3030	788.0%	165	829	402.9%
	131073	967	3842	297.1%	697	686	-1.7%
	262145	381	3670	863.8%	214	339	58.7%
	524289	1144	4174	265.0%	620	525	-15.3%
	1048577	177	2743	1453.5%	113	244	116.9%
	2097153	1125	3747	233.0%	617	552	-10.6%
	4194305	314	2063	557.5%	175	394	124.9%
	8388609	632	2628	315.5%	292	409	40.1%
Average				593.38%			96.65%

Table 9. Results for many short cycles graphs SQNC03

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
SQNC03	4097	109	487	345.3%	55	173	216.1%
	16385	74	584	686.5%	34	249	625.0%
	65537	250	1101	340.5%	130	493	280.1%
	262145	477	3779	691.6%	243	1077	343.2%
	1048577	1818	7839	331.1%	950	2441	157.0%
	4194305	2934	7534	156.8%	1536	4593	199.0%
	16777217	5896	28282	379.7%	3066	8719	184.4%
Average				418.79%			286.40%

Table 10. Results for a few long cycles graphs LNC04

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
LNC04	8193	26711	54962	105.8%	12910	20687	60.2%
	16385	92773	148990	60.6%	41107	56489	37.4%
	32769	251488	349870	39.1%	122379	145197	18.6%
	65537	881848	948447	7.6%	365853	376227	2.8%
	131073	1980993	2097090	5.9%	874778	898293	2.7%
	262145	4467623	5460818	22.2%	1892207	2215706	17.1%
	524289	11883080	11385873	-4.2%	4968220	5017279	1.0%
	1048577	25741146	28462486	10.6%	10792331	12567955	16.5%
	2097153	62102224	61088246	-1.6%	26089374	27892978	6.9%
	4194305	121983730	128064383	5.0%	53771346	59798591	11.2%
	8388609	275905623	286903304	4.0%	112925426	135532732	20.0%
	16777217	538729562	664472322	23.3%	231721552	299999324	29.5%
Average				23.19%			18.66%

Table 11. Results for a few long cycles graphs PNC04

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
PNC04	8193	96459	82610	-14.4%	55914	37737	-32.5%
	16385	363660	189675	-47.8%	222163	89015	-59.9%
	32769	1032742	416393	-59.7%	633831	215134	-66.1%
	65537	2475918	932459	-62.3%	1574038	485710	-69.1%
	131073	3959249	2244526	-43.3%	2482065	1042791	-58.0%
	262145	11673063	4881046	-58.2%	6672867	2282332	-65.8%
	524289	26999818	10280214	-61.9%	16888696	4707412	-72.1%
	1048577	56517150	21103100	-62.7%	35114982	9594361	-72.7%
	2097153	118365443	44260038	-62.6%	74302031	20881883	-71.9%
	4194305	225343295	95936613	-57.4%	147809535	43256228	-70.7%
	8388609	517666048	189688611	-63.4%	328137483	88085199	-73.2%
Average				-53.97%			-63.74%

Table 12. Results for a few long cycles graphs SQNC04

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
SQNC04	4097	32707	30880	-5.6%	15213	13944	-8.3%
	16385	184893	170162	-8.0%	81463	67445	-17.2%
	65538	573055	746841	30.3%	254273	297501	17.0%
	262145	1313928	3264595	148.5%	560271	1389863	148.1%
	1048577	2338506	15068659	544.4%	1099832	6288913	471.8%
	4194305	6204248	68242481	999.9%	2710118	27430729	912.2%
	16777217	14024018	272653950	1844.2%	6681120	121993247	1725.9%
Average				507.67%			464.21%

Table 13. Results for Hamiltonian cycle graphs LNC05

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
LNC05	8193	163438	172029	5.3%	98947	91658	-7.4%
	16385	346868	364541	5.1%	215632	202262	-6.2%
	32769	753945	803979	6.6%	458880	436855	-4.8%
	65537	1633918	1733490	6.1%	999207	934728	-6.5%
	131073	3499646	3789556	8.3%	2082565	2027643	-2.6%
	262145	7599126	7828071	3.0%	4637622	4329335	-6.6%
	524289	15608355	16876480	8.1%	9456863	9179454	-2.9%
	1048577	33572684	35232411	4.9%	20445301	19287842	-5.7%
	2097153	69453620	74266615	6.9%	42213747	40477734	-4.1%
	4194305	144322754	156777150	8.6%	89613042	84456141	-5.8%
	8388609	314273826	331798694	5.6%	189346386	181584841	-4.1%
	16777217	637684719	668256739	4.8%	387236035	364980712	-5.7%
Average				6.11%			-5.20%

Table 14. Results for Hamiltonian cycle graphs PNC05

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
PNC05	8194	82594	90718	9.8%	54836	47319	-13.7%
	16386	179350	203306	13.4%	116631	100007	-14.3%
	32770	361404	401511	11.1%	249244	210670	-15.5%
	65538	785227	870523	10.9%	516925	443592	-14.2%
	131074	1732688	1680790	-3.0%	1096942	901936	-17.8%
	262146	3462811	3965235	14.5%	2232233	1928211	-13.6%
	524290	7714697	7216528	-6.5%	4579138	4039317	-11.8%
	1048578	16872712	16794782	-0.5%	9786195	8450277	-13.7%
	2097154	32924136	37489022	13.9%	19973817	17243048	-13.7%
	4194306	63798970	71903161	12.7%	41054033	35357906	-13.9%
	8388610	154010041	151672136	-1.5%	86223637	74532047	-13.6%
Average				-1.50%			-14.16%

Table 15. Results for Hamiltonian cycle graphs SQNC05

	#Vertices	MGORC	GORC	Improve	MBFCT	BFCT	Improve
SQNC05	4098	70976	79742	12.4%	44744	41610	-7.0%
	16386	360423	365960	1.5%	207067	197097	-4.8%
	65538	1644198	1712316	4.1%	983910	928755	-5.6%
	262146	7650803	8070454	5.5%	4679524	4297402	-8.2%
	1048578	32523850	34451433	5.9%	19818466	18737042	-5.5%
	4194306	151666794	149113446	-1.7%	91909813	82151343	-10.6%
	16777218	648487235	648211387	0.0%	394621759	353276864	-10.5%
Average				3.96%			-7.46%

Table 16. Results for random graphs RAND-5

Negative Weight	MGORC	GORC	Improve	MBFCT	BFCT	Improve
0	0	7780968	INF	0	4578450	INF
-1000	1133358	8892373	684.6%	341958	5350722	1464.7%
-2000	2080317	9588453	360.9%	762572	6481434	749.9%
-4000	260253	3665786	1308.5%	211730	5139325	2327.3%
-6000	328345	884558	169.4%	218531	1634924	648.1%
-8000	224668	41588	-81.5%	115428	721280	524.9%
-16000	603	541	-10.4%	328455	237747	-27.6%
-32000	175	174	-0.5%	329102	143538	-56.4%
-64000	51	48	-4.3%	380534	55175	-85.5%

Table 17. General results for GORC

	Improve	Degrade
$100\% < X$	36.1%	0.0%
$10\% < X \leq 100\%$	27.8%	17.0%
$X \leq 10\%$	12.8%	6.3%
Total	76.7%	23.3%

Table 18. General results for BFCT

	Improve	Degrade
$100\% < X$	24.8%	0.0%
$10\% < X \leq 100\%$	29.3%	25.7%
$X \leq 10\%$	9.6%	10.6%
Total	63.7%	36.3%

Appendix II

Proof of Lemma 1

Consider a negative cost cycle C in a directed graph $G(V, E)$ where C=$\{v_0, v_1, \ldots, v_{n-1}\}$. The accumulated costs on C starting from v_0 are $\{C_{1,0}, C_{1,1}, \ldots, C_{1,n-1}\}$, where $C_{1,k}$ is the accumulated cost at vertex v_k starting from v_0.

First of all, at the end of the cycle, i.e. v_{n-1}, the accumulated cost (W_{cycle}) along this cycle must be negative. Secondly, there is at least one maximum accumulated cost C_{max} on this cycle. Finally, there must be at least one edge, (v_i ,v_j), where $j = i$+1, that is negative since there must be at least one negative edge on a negative cycle. Note that we can change the ordering of the vertices by choosing another "starting vertex". Suppose that we take v_k as the starting vertex where v$_k$ has the largest accumulated cost (chose the last one if there is more than one), we will have the accumulated costs $\{C_{2,k+1}, C_{2,k+2}, \ldots, C_{2,n-1}, C_{2,0}, C_{2,1}, \ldots, C_{2,k}\}$.

Now we will have the path $\{v_{k+1}, v_{k+2}, \ldots, v_{n-1}, v_0, v_1, \ldots, v_k\}$. Then we will have two sets of vertices. The first set contains the vertices in $\{v_{k+1}, v_{k+2}, \ldots, v_{n-1}\}$. For any vertex v in this set, $C_2,v = C_1,v - C_{max} < 0$, where C_{max}≧0 because otherwise, v_0 is already the correct starting point. The second set contains the vertices in $\{v_0, v_1, \ldots, v_k\}$. For any vertex v in this set, $C_2,v = C_1,v + W_{cycle} - C_{max} < 0$.