JA-BE-JA

Fernando Díaz **Giorgio Ruffa**

{fdiaz, ruffa}@kth.se

1 Task 1: implementation

We modified three methods: `findParter`, `sampleAndSwap` and `saCoolDown`. Keep in mind that we modified the last method to allow using different annealing policies (the mechanism explained in the paper or the one required for task 2), and to allow resetting the temperature after it reaches its minimum value. We attach a listing of the code in Sec. 4.

To test the implementation, we show the results for three different graphs using 2 different node selection policies: `LOCAL` and `HYBRID`. The values for the initial temperature, alpha and delta are the default ones ($T = 2, \alpha = 2, \delta = 0.003$).

The results with this configuration were:

Graph	Policy	Ratio cut	Swaps	Migrations
3elt	LOCAL	3209	1599914	3346
3elt	HYBRID	2604	1580209	3328
add20	LOCAL	3228	1536807	1645
add20	HYBRID	2313	2393224	1747

Table 1: Node selection policy comparison. Using simulated annealing (as described in the paper) with $T = 2, \alpha = 2$ and $\delta = 0.003$

.

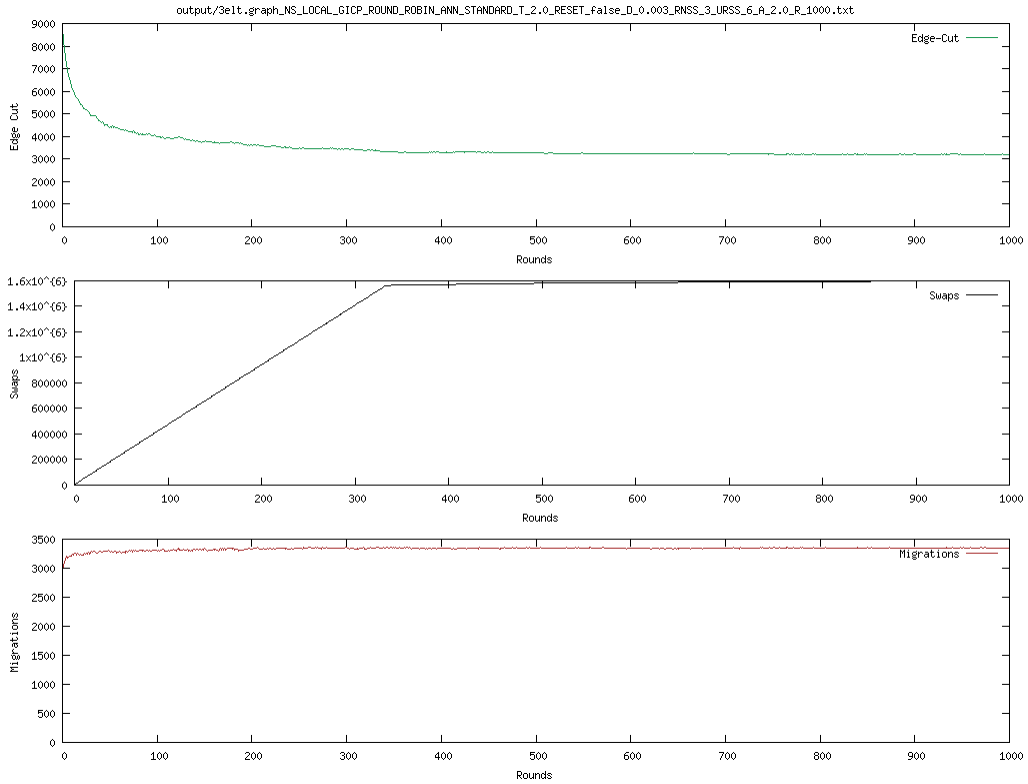


Figure 1: 3elt graph (*LOCAL* policy) edge cut: 3209, swaps: 1599914, migrations: 3346

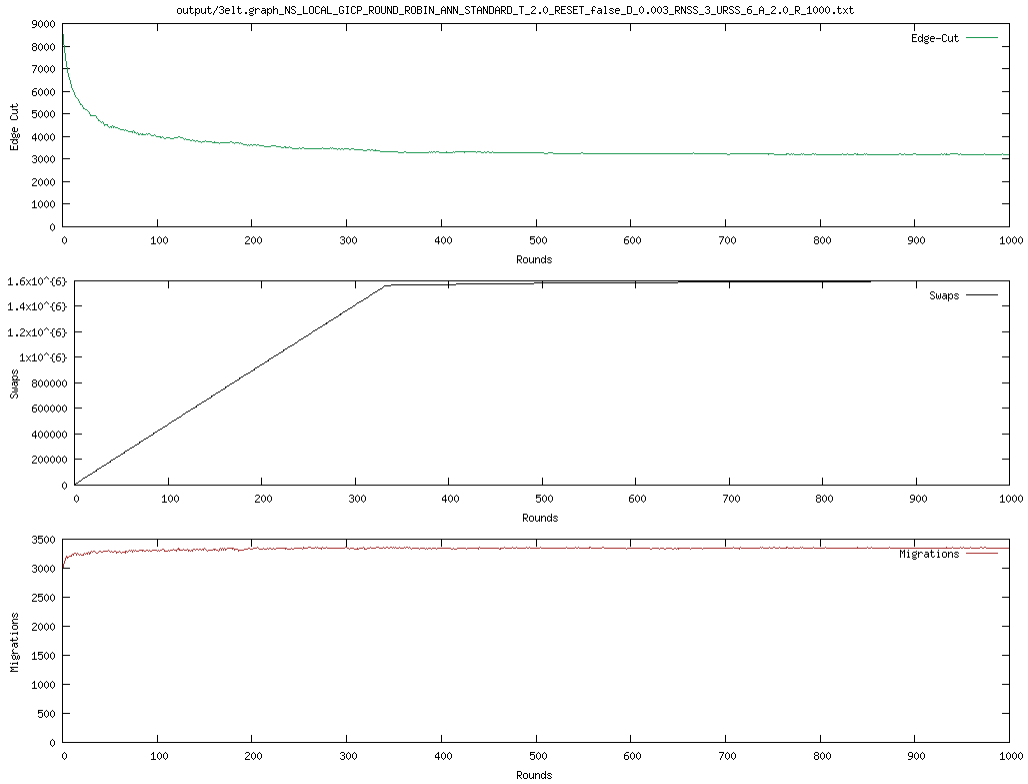


Figure 2: 3elt graph (*HYBRID* policy) edge cut: 2604, swaps: 1580209, migrations: 3328

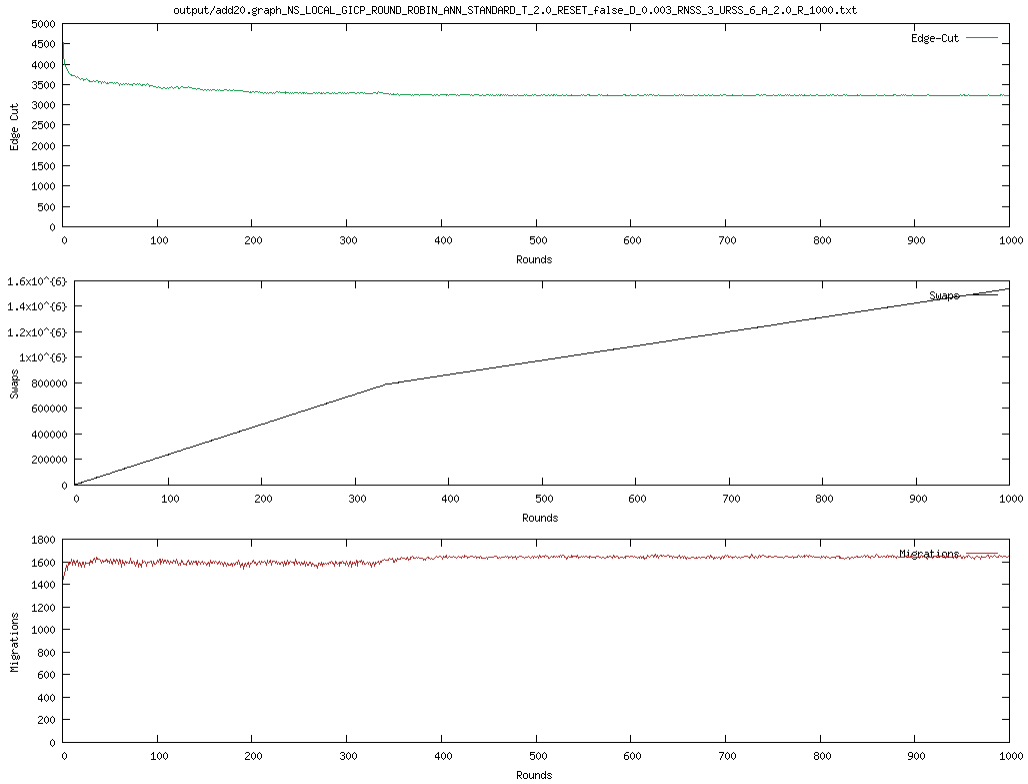


Figure 3: add20 graph (*LOCAL* policy) edge cut: 3228, swaps: 1536807, migrations: 1645

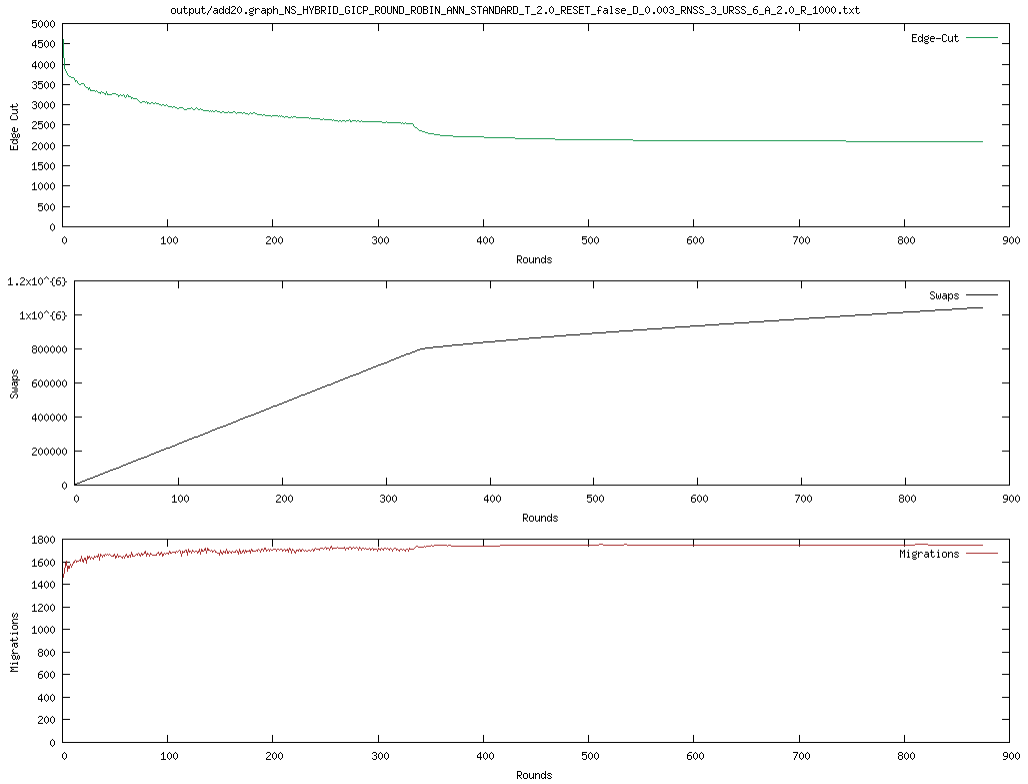


Figure 4: add20 graph (*HYBRID* policy) edge cut: 2313, swaps: 2393224, migrations: 1747

2 Task 2: exponential annealing

For the exponential annealing, given the `oldScore` (graph unchanged) and the `newScore` (pair of nodes swapped), we calculate an acceptance probability as follows:

$$a = e^{\frac{new-old}{T}}$$

Is it easy to see that a get smaller as the new score is lower than the old score. Also, as the temperature increases, we penalize bad jumps (e.g., $e^{\frac{-1}{1}} > e^{\frac{-1}{0.5}}$).

The results obtained were:

Graph	Delta	Ratio cut	Swaps	Migrations
3elt	0.9	1994	26617	3320
3elt	0.7	1645	34061	3339
add20	0.9	1972	614550	1719
add20	0.7	1779	524018	1644

Table 2: Effect of δ on the exponential annealing. Using simulated annealing (with acceptance probability) with $T = 1, \alpha = 2$ and different δ's

.

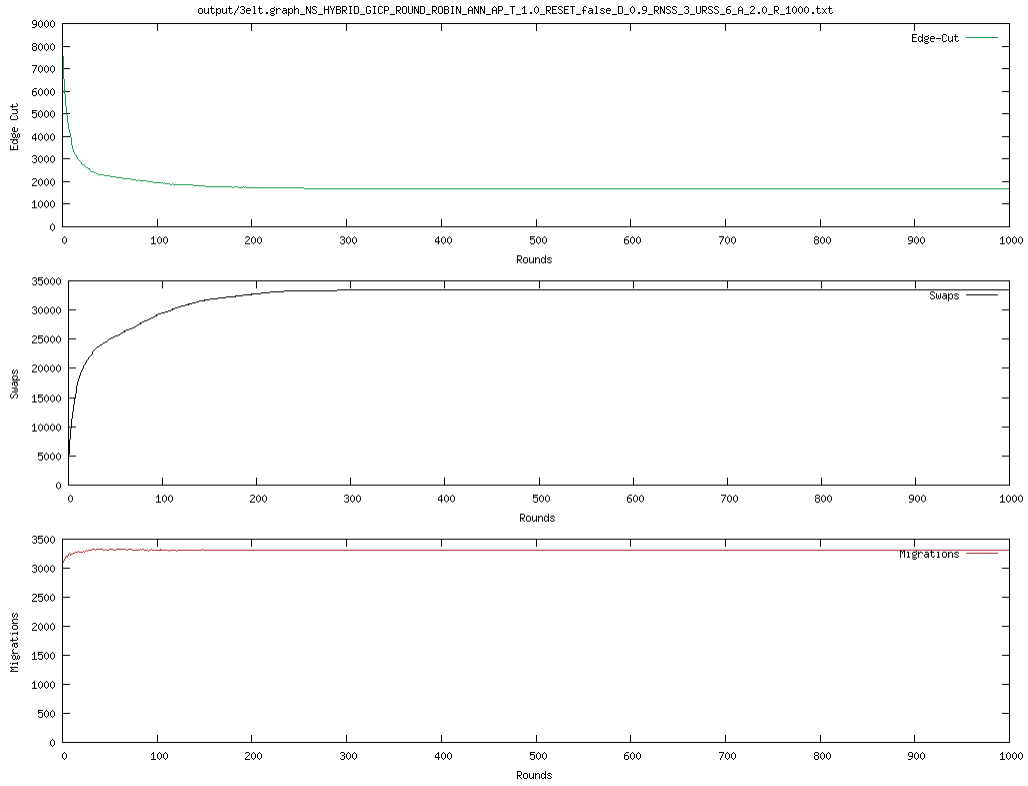


Figure 5: 3elt graph (EXP AP, $\delta = 0.9$) - edge cut: 1994, swaps: 26617, migrations: 3320

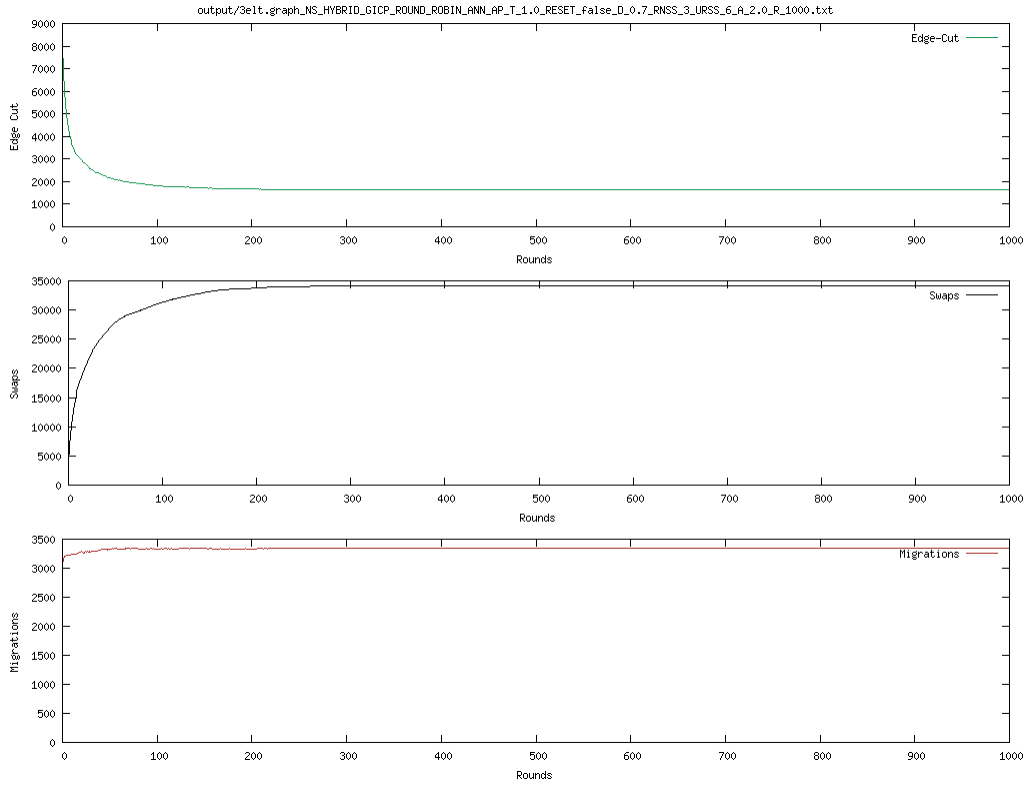


Figure 6: 3elt graph (EXP AP, $\delta = 0.7$) - edge cut:1645, swaps: 34061, migrations: 3339

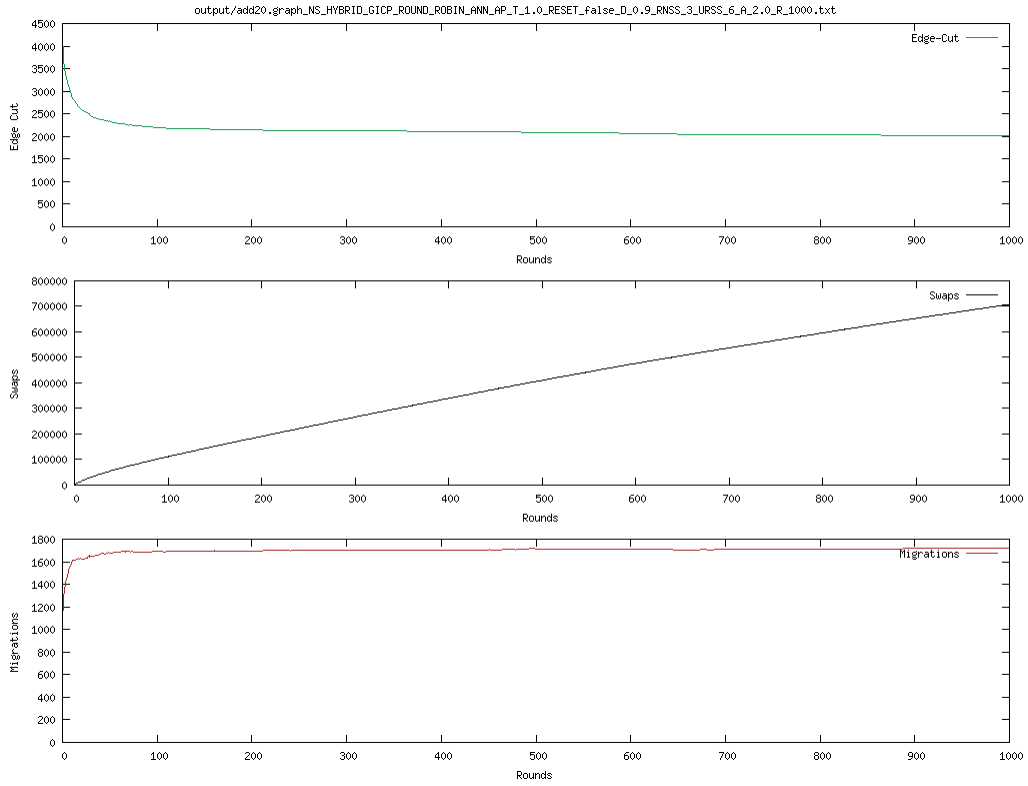


Figure 7: add20 graph (EXP AP, $\delta = 0.9$) - edge cut: 1972, swaps: 614550, migrations: 1719

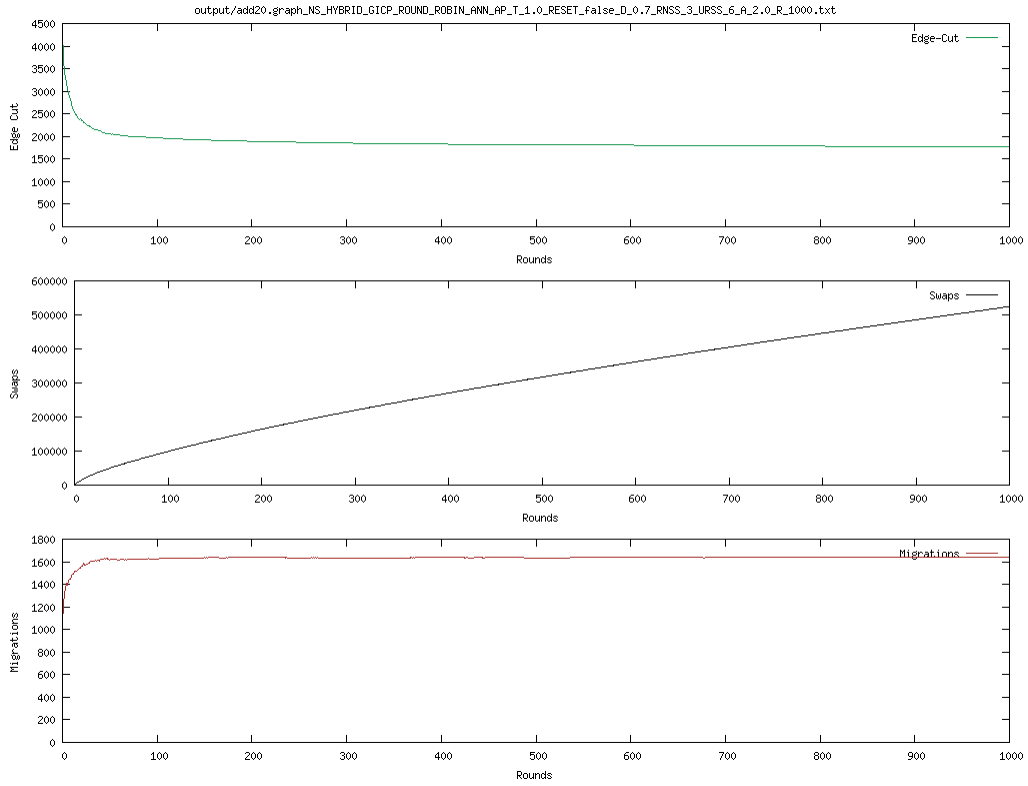


Figure 8: add20 graph (EXP AP, $\delta = 0.7$) - edge cut: 1779, swaps: 524108, migrations: 1644

2.1 Resetting T

In this section, we perform the same experiments than in the previous section (exponential annealing), but we reset T 400 rounds after it reaches its minimum value. The results are:

Graph	Annealing	Delta	Ratio cut	Swaps	Migrations
3elt	AP	0.9	1993	27627	3328
3elt	STANDARD	0.003	2608	2839724	3364
add20	AP	0.9	2009	549496	1794
add20	STANDARD	0.003	2253	1628414	1742

Table 3: Resetting T. Using simulated annealing (STANDARD and AP) with default values for $T\alpha$ and δ

.

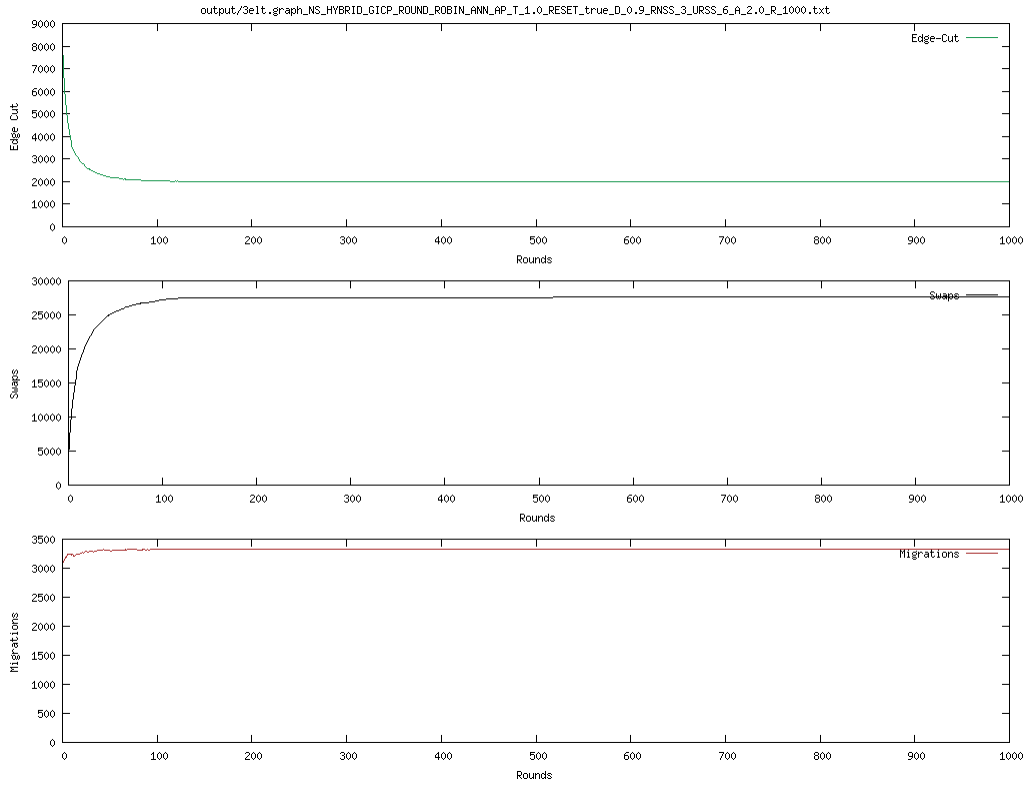


Figure 9: 3elt graph (EXP AP, $\delta = 0.9$, RES) - edge cut: 1993, swaps: 27627, migrations: 3328

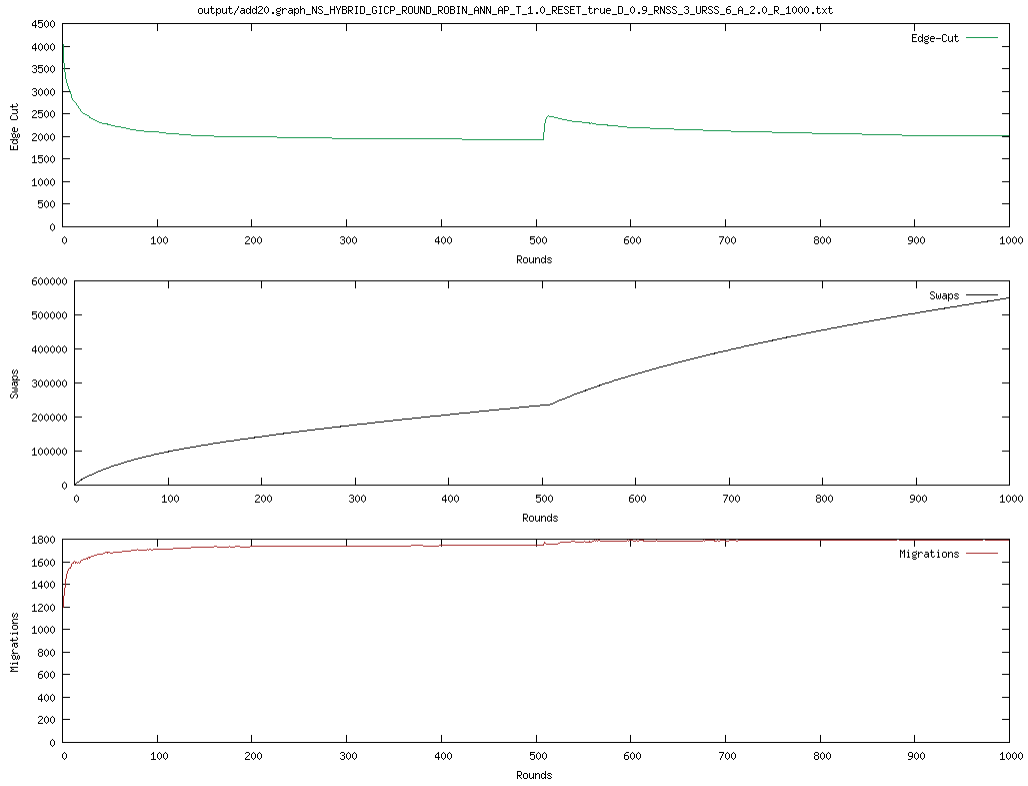


Figure 10: add20 graph (EXP AP, $\delta = 0.9$, RES) - edge cut: 2009, swaps: 549496, migrations: 1794

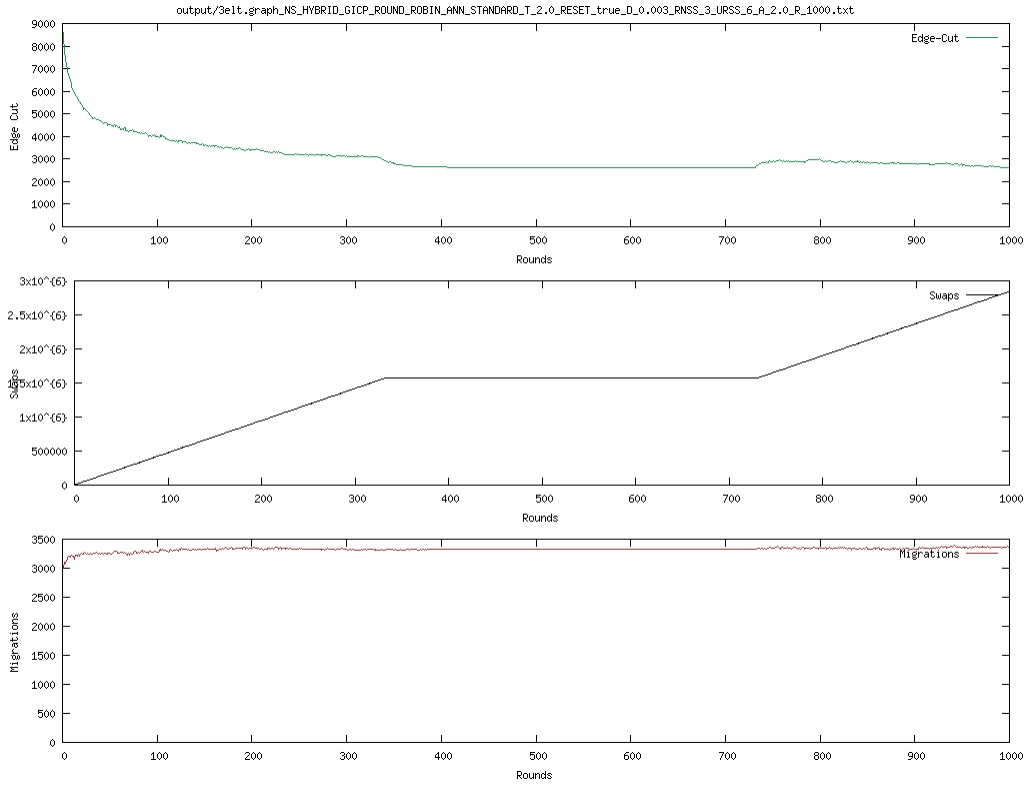


Figure 11: 3elt graph (STD, RES) - edge cut: 2608, swaps: 2839724, migrations: 3364

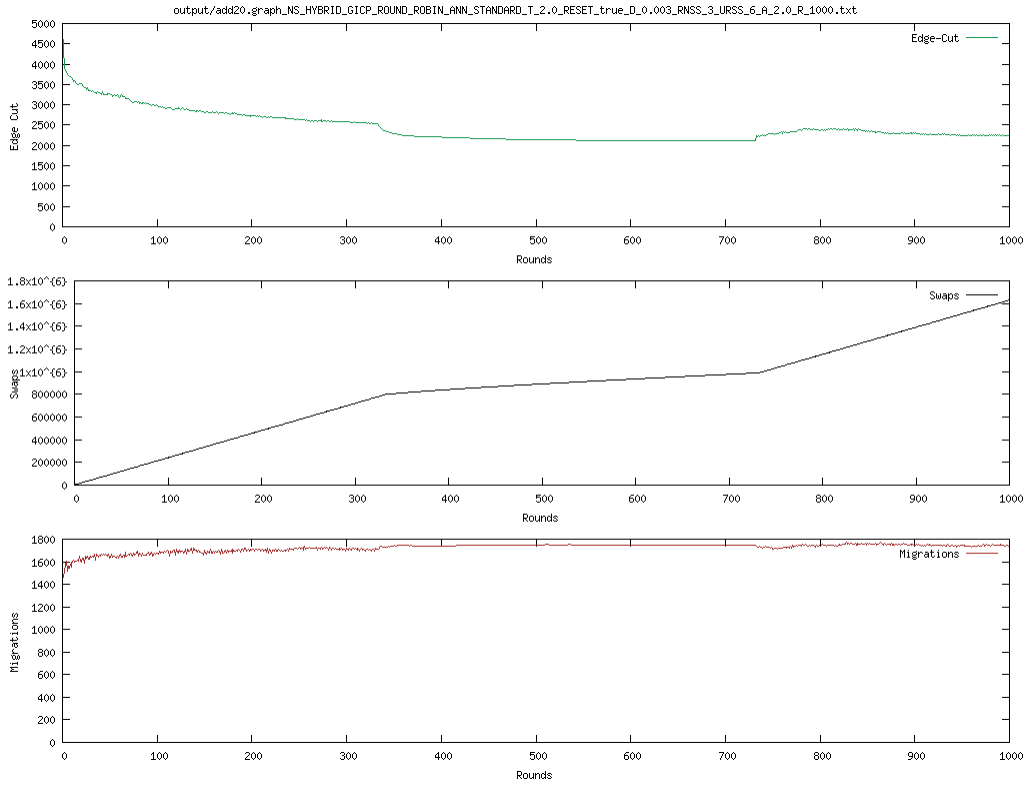


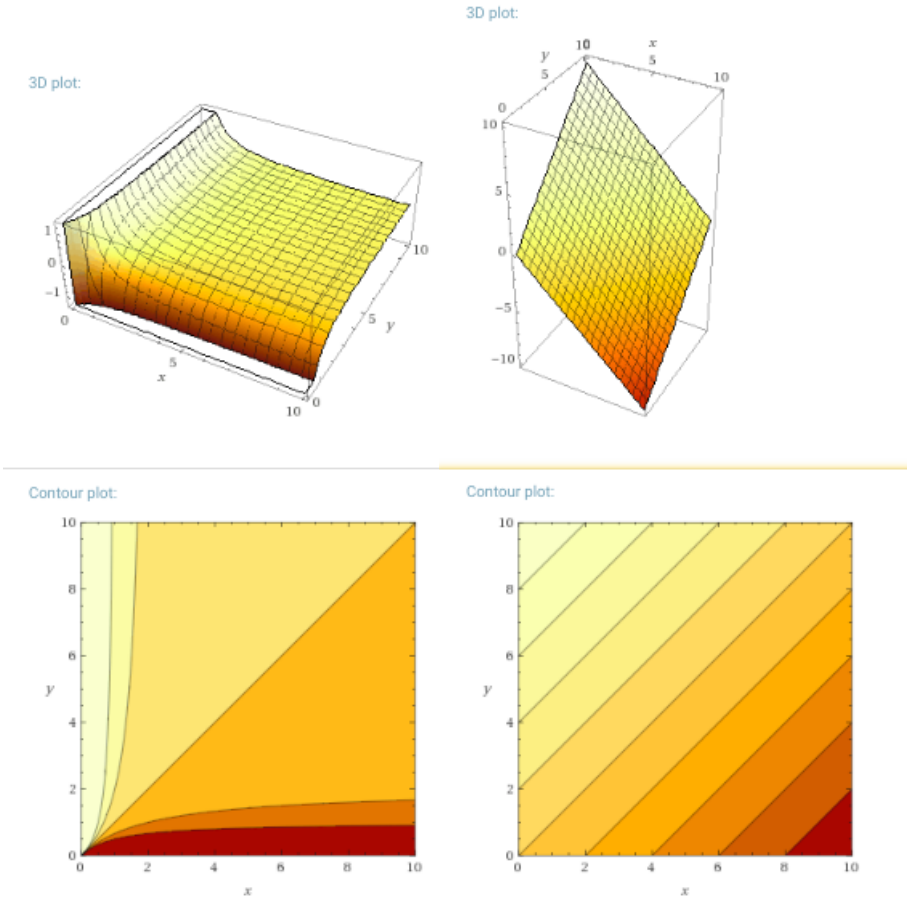
Figure 12: 3elt graph (STD, RES) - edge cut: 2253, swaps: 1628414, migrations: 1742

3 Task 3: harmonic difference AP

We use the following equation:

$$a = e^{\frac{\frac{1}{old} - \frac{1}{new}}{T}} \tag{1}$$

The difference between this equation and the original one can be derived for the following figures



(a) $f(x, y) = \frac{1}{y} - \frac{1}{x}$ (b) $f(x, y) = x - y$

Figure 13: Comparison between harmonic difference (left) and standard difference (right)

As you can see the harmonic difference grants an wider surface where the argument of the exponential is close to 0, thus yielding a probability close to 1. The argument quickly diverges to $-\infty$ for high differences between the scores. We expect a higher number of swaps when the temperature is still high.

The results were:

Graph	Annealing	Delta	Ratio cut	Swaps	Migrations
3elt	CUSTOM	0.9	1496	217545	3573
add20	CUSTOM	0.9	1873	868556	1806

Table 4: Custom acceptance probability (harmonic-difference) with default values for $T\alpha$ and δ

.

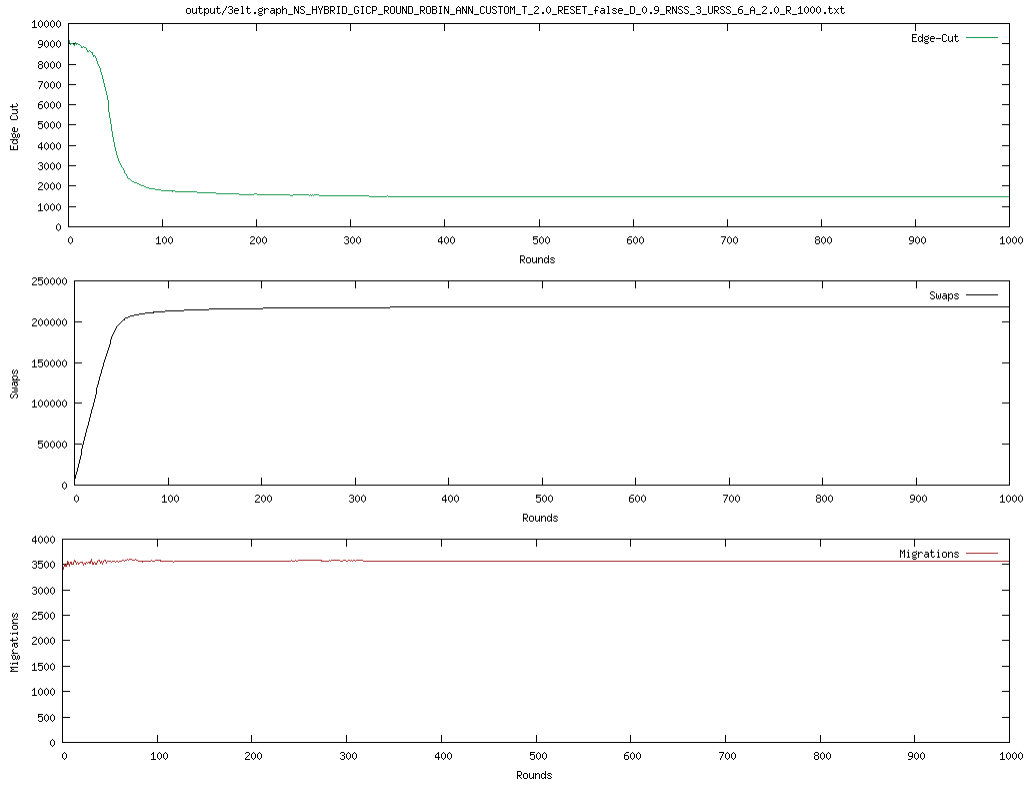


Figure 14: 3elt graph (CUSTOM AP, $\delta = 0.9$) - edge cut: 1496, swaps: 217545, migrations: 3573

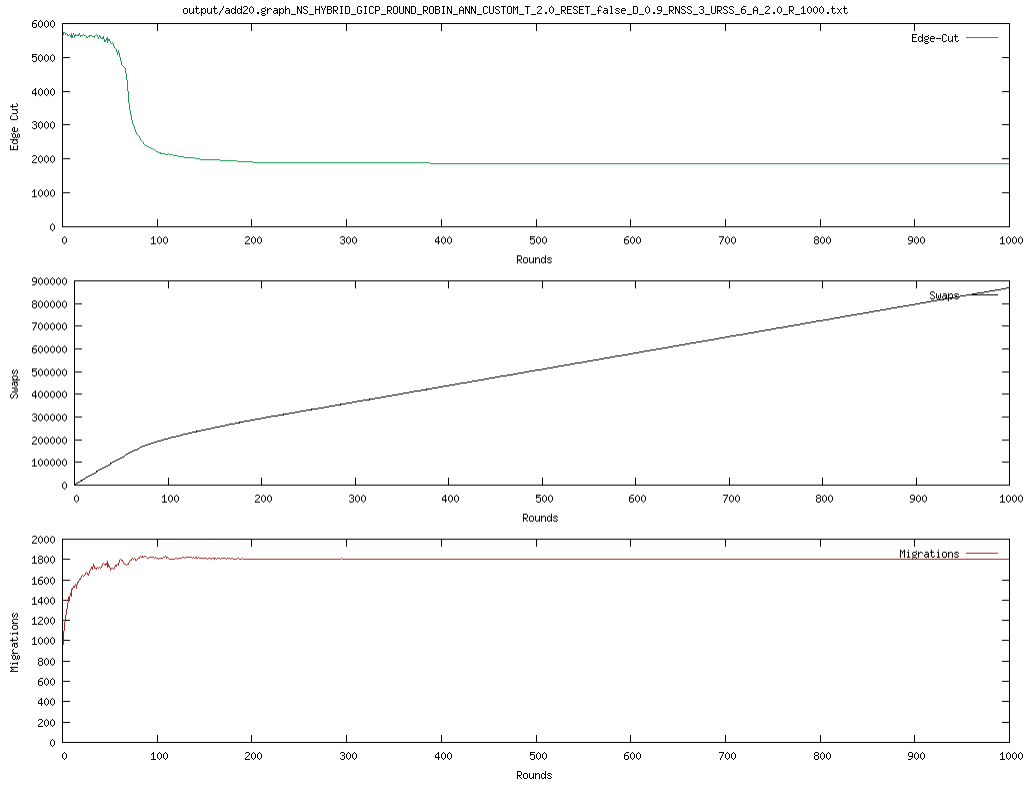


Figure 15: add20 graph (CUSTOM AP, $\delta = 0.9$) - edge cut: 1873, swaps: 868556, migrations: 1806

4 Appendix: code

```
1    private void saCoolDown() {
2        if (T > T_min) {
3            if (config.getAnnealingPolicy() == AnnealingPolicy.
                 STANDARD) {
4                T -= config.getDelta();
5            } else if (config.getAnnealingPolicy() == AnnealingPolicy.
                 EXP
6                    || config.getAnnealingPolicy() == AnnealingPolicy.
                         CUSTOM) {
7                T *= config.getDelta();
8            }
9            logger.info("Cooling T to " + T);
10       }
11       if (T < T_min) {
12           T = T_min;
13           this.roundsWaitingToReset = 0;
14       }
15       if (T == T_min && this.config.shouldResetT()) {
16           this.roundsWaitingToReset++;
17           int roundsRemaining = ROUNDS_BEFORE_RESET - this.
                 roundsWaitingToReset;
18           logger.info("Waiting " + (roundsRemaining) + " more rounds
                  to reset T");
19           if (this.roundsWaitingToReset == ROUNDS_BEFORE_RESET) {
20               this.T = this.config.getTemperature();
21               this.roundsWaitingToReset = 0;
22           }
23       }
24   }
25
26       private void sampleAndSwap(int nodeId) {
27       Node partner = null;
28       Node nodep = entireGraph.get(nodeId);
29
30       if (config.getNodeSelectionPolicy() == NodeSelectionPolicy.
             HYBRID
31               || config.getNodeSelectionPolicy() ==
                     NodeSelectionPolicy.LOCAL) {
32           // swap with random neighbors
33           partner = this.findPartner(nodeId, this.getNeighbors(nodep
                 ));
34           if (partner != null)
35               counterLocal++;
36       }
37
38       if (config.getNodeSelectionPolicy() == NodeSelectionPolicy.
             HYBRID
39               || config.getNodeSelectionPolicy() ==
                     NodeSelectionPolicy.RANDOM) {
40           // if local policy fails then randomly sample the entire
                 graph
41           if (partner == null) {
42               partner = this.findPartner(nodeId, this.getSample(
                     nodeId));
43               if (partner != null)
44                   counterRandom++;
45           }
46       }
47
48       // swap the colors
49       if (partner != null) {
50           int partnerColor = partner.getColor();
51           partner.setColor(nodep.getColor());
```

```
52              nodep.setColor(partnerColor);
53              this.numberOfSwaps++;
54          }
55      }
56
57      public Node findPartner(int nodeId, Integer[] nodes) {
58
59          Node nodep = entireGraph.get(nodeId);
60
61          Node bestPartner = null;
62          double highestBenefit = 0;
63
64          for (Integer nodeqId : nodes) {
65              Node nodeq = entireGraph.get(nodeqId);
66              int dpp = this.getDegree(nodep, nodep.getColor());
67              int dqq = this.getDegree(nodeq, nodeq.getColor());
68              double oldScore = pow(dpp, this.alpha) + pow(dqq, this.
                    alpha);
69              int dpq = this.getDegree(nodep, nodeq.getColor());
70              int dqp = this.getDegree(nodeq, nodep.getColor());
71              double newScore = pow(dpq, this.alpha) + pow(dqp, this.
                    alpha);
72
73              if (config.getAnnealingPolicy() == AnnealingPolicy.
                    STANDARD) {
74                  if (((newScore * this.T) > oldScore) && (newScore >
                        highestBenefit)) {
75                      bestPartner = nodeq;
76                      highestBenefit = newScore;
77                  }
78              } else if (config.getAnnealingPolicy() == AnnealingPolicy.
                    EXP) {
79                  double ap = Math.exp((newScore - oldScore) / T);
80                  double ran = RANDOM.nextDouble();
81                  if (newScore != oldScore && ap > ran && ap >
                        highestBenefit) {
82                      bestPartner = nodeq;
83                      highestBenefit = ap;
84                  }
85              } else if (config.getAnnealingPolicy() == AnnealingPolicy.
                    CUSTOM) {
86                  double ap = Math.exp((1 / oldScore - 1 / newScore) / T
                        );
87                  double ran = RANDOM.nextDouble();
88                  if (newScore != oldScore && ap > ran && ap >
                        highestBenefit) {
89                      bestPartner = nodeq;
90                      highestBenefit = ap;
91                  }
92              }
93
94          }
95
96          return bestPartner;
97      }
```