Parallel Simulated Annealing Algorithm Study

Brito, Rui
PG22781
Department of Informatics
University of Minho
ruibrito666@gmail.com

Alves, José
PG22765
Department of Informatics
University of Minho
zealves.080@gmail.com

Abstract

1 Introduction

In this report we present a study of a Parallel Simulated Annealing algorithm for solving the room assignment problem. Simulated annealing is an iterative method based on the process of physical annealing. This algorithm starts by choosing a solution which might be worse than a solution given by a probability function. In each iteration it decrease the probability of a worse solution, closing the gap from a local limited solution to a more global and optimized one.

This project compares two algorithms for solving the Room Assignment problem, one with Simulated Annealing and one without. The algorithms were both implemented in a serial fashion and in openMPI. Both versions were tested with the same matrix of incompatibility values from a range of pseudo-generated values.

This report studies the result of the simulated annealing method in the Room Assignment problem, focusing the variation of parameters and the results obtained.

This report starts with a brief Introduction followed by a Case Study section where the problem is explained. The Algorithm section shows the implementation of said algorithm and some reasoning behind some decisions. We then present the results, comparing both versions of the algorithm in the section Results. Finally a brief conclusion summarizes the project in the last section of the report.

2 Case Study

The Case Study here presented is the Room Assignment Problem. This is a problem of optimization where the simulated annealing gives better solutions than a regular probabilistic algorithm.

The problem consists in assigning students(n) to rooms($\frac{n}{2}$), but minimizing the conflicts between roommates.

The objective of this case study is to minimize the cost of the distribution. Since the magnitude of the problem is very big and there is no way of analyzing all possible states in a acceptable amount of time, we use simulated annealing. Using this method we can make a good approximation of the optimal value with less computation.

To start, a pseudo-random matrix is created to represent the room distribution. For each iteration two randomly selected students are exchanged, if the cost is lower in the new matrix-state, the change is made, if not, it is reversed. The conflicts are registered in a matrix, where it shows how many students have conflicts with each other. The current cost of a state is equal do the sum of all conflicts.

The method sets a high temperature for the system, making the first iterations more disperse and random. While the temperature of the system lowers with each iteration, the result starts to approach a better solution, finding a better global solution with lower cost.

The method of simulated annealing modifies the conditions of how the swap is taken. Using simulated annealing the system considers the difference between the state before the swap and after the swap. Without simulated annealing only states with lower cost are accepted making the solution more narrow and local.

3 Algorithm

The program developed has a sequential version and a distributed memory version. The first part of the algorithm maintains the same structure, first selecting two students who are not roommates, find their

respective rooms and calculating their conflict ratio.

```
    (...)
//Select students
s1 = randomul_limited(0, laststudent
    );
s2 = randomul_limited(0, laststudent
    );
(...)

// Room of the first student
r1 = assigned[s1];
sa = rooms[r1 * 2    ];
sb = rooms[r1 * 2 + 1];
p1 = (s1 == sa);
s3 = p1 ? sb : sa;

// Room of the second student
r2 = assigned[s2];
sa = rooms[r2 * 2    ];
sb = rooms[r2 * 2 + 1];
p2 = (s2 == sa);
s4 = p2 ? sb : sa;

// Conflit coeficient
dcost = dislikes[s1 * nstudents + s4
    ]
              + dislikes[s2 *
                  nstudents + s3]
              - dislikes[s1 *
                  nstudents + s3]
              - dislikes[s2 *
                  nstudents + s4];
```

In the next step is where the algorithms differ. The algorithm below shows what is done for the version without simulated annealing.

```
        if ( dcost < 0 ) {
             assigned[s3] = r2;
             assigned[s4] = r1;
             rooms[r1 * 2 + p1] = s4;
             rooms[r2 * 2 + p2] = s3;
             if (dcost) {
                 cost += dcost;
                 j = nstudents;
             } else
                 --j;

             i = max;
         }
         else
         {
             --i;
         }
```

As we can see above, the algorithm only accepts change when the dislike cost of each specific room is lower than before. This makes the solution very local since it doesn't take into consideration the other rooms.

```
        double r = randDouble();
        double f = (double) (- dcost
            ) / (double) t;
        double e = exp(f);

        if (dcost < 0 || (e >= r)) {
            assigned[s3] = r2;
            assigned[s4] = r1;
            rooms[r1 * 2 + p1] = s4;
            rooms[r2 * 2 + p2] = s3;

            if (dcost) {
                cost += dcost;
                j = nstudents;
            } else
                --j;

            if (cost && dcost < 0 &&
                 ((unsigned long)(-
                dcost)) > cost)
                cost = 0;

            i = max;
        }
        else {
            --i;
        }

        //cool the system
            t *= 0.999;
    }
```

In the simulated annealing version the algorithm compares the swapping cost with the current one, if the swap cost is lower or the temperature is high enough so that $e^{-dcost/T}$ is higher than a randomly generated value, r, the swap is accepted and the number of iterations set to zero.

One of the key components in this algorithm is precisely the condition of acceptance based on the `exp` function. This condition, meant to decide whether a worse state will be accepted or not, always causes a state with equal cost to be accepted. Despite not changing the cost, rearranging some students may unlock better combinations in the next iterations, which makes this a desirable feature. At the same time, in late stages of the execution, when the temperature is so low that only the better solutions are accepted, this may cause the algorithm to follow loops, constantly resetting the stagnant counter and extending the execution time beyond usable wall times. To correct this, a second iteration counter is added, which is reset only when the cost is changed. While this new counter allows the

algorithm to keep accepting states with the same cost, it forces the stop when too many iterations have passed with no improvements.

4 Results

First we can observe the results for the serial version of the algorithm. Two different random number generators were used, `rand` and `arc4random` (available in BSD and its descendants). In both cases, the `rand` generator succeeded in achieving the perfect solution more times than the `arc4random`. Yet, the `arc4random` generator follows a very low and constant line of results, while `rand` is very irregular in the mid case, where it obtained values near the sequential results when it did not manage to find the perfect solution.

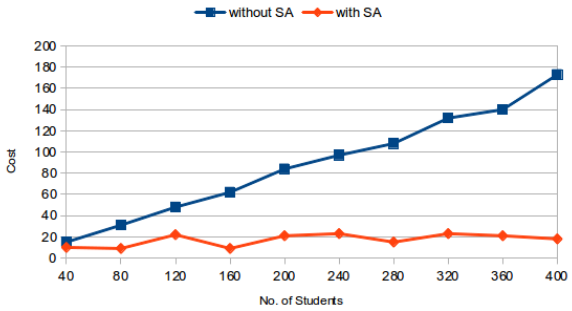

Figure 1: Serial execution using the rand() generator

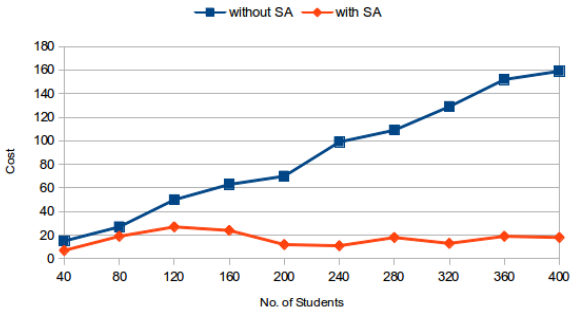

Figure 2: Serial execution using the arc4random() generator

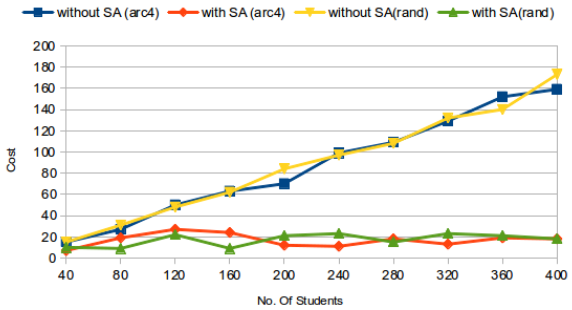

Figure 3: Comparison of both generators

As shown above, the advantage in using simulated annealing is clear. By letting the algorithm follow worse solutions in the beginning, it achieved values around 78% better (average) than those of the basic version. In fact, in the sequential version, the basic approach never even obtained the perfect solution.

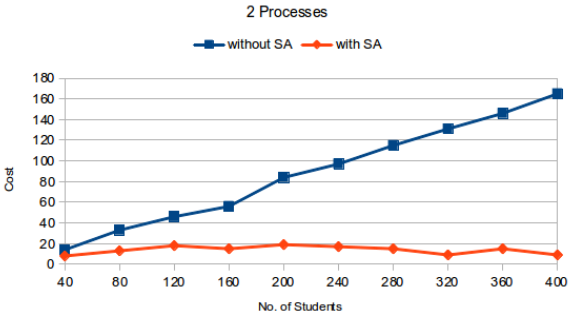

Figure 4: Results for the program in 2 Processes.

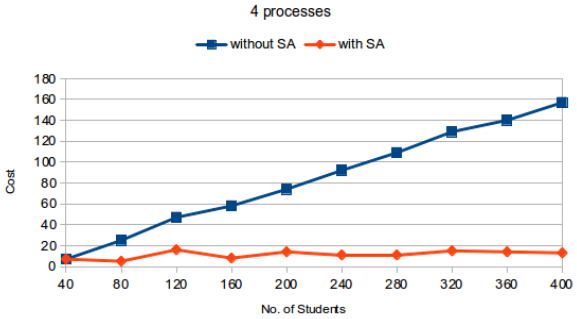

Figure 5: Results for the program in 4 Processes.

As we can see in the results, the final solution is always better for the simulated annealing approach. Taking advantage of parallelism, therefore allowing more solutions to be computed at the same time introduced even greater improvements. But even in the largest test the basic approach barely touched the perfect solutions. The simulated annealing ap-

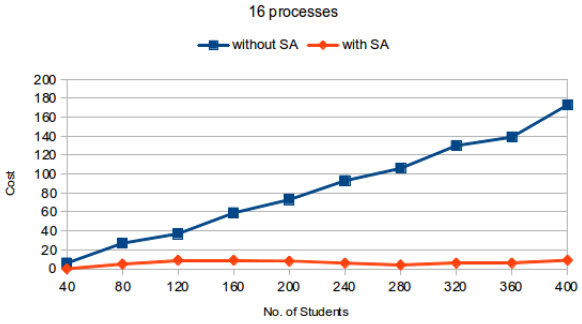

Figure 6: Results for the program in 16 Processes.

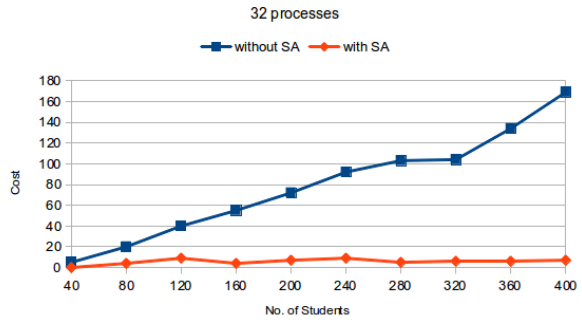

Figure 7: Results for the program in 32 Processes.

proach, on the other hand, almost obtained the perfect solution for 16 processes or more. Compared with the basic approach in the sequential version, the improvement with simulated annealing and using 16 processes is around 94% (average).

The results clearly show a improvement in the solutions when using simulated annealing. Following a worse case solution in the first iterations, led us to find a better overall solutions. Since the algorithm is embarrassingly parallel the costs of parallelizing the application were minimal and provide us with more resources to find a better solution in a similar amount of time.

5 Conclusion

In this document, the simulated annealing method was studied using the Room Assignment problem. Since the problem is too computationally heavy to solve deterministically, an heuristic method was used.

As stated in the previous section, the algorithm is embarrassingly parallel, so an approach using distributed memory was created. Since each process can calculate its optimal solution using their pseudo generated numbers, a better result can be achieved by the program.

Both versions were tested with different numbers of students. Simulated annealing was proved to greatly improve the results of the method, being this effect amplified in the parallel version. Regarding the variation of temperature, variations using a few hundreds of students did not influence the results noticeably. Yet, analyzing the first iterations, those same tests prove the textbook's claims that increasing the initial temperature slows the convergence of the method.

The final results were objective, showing much better cost for the simulated annealing approach. Allowing the algorithm to follow worse solutions in the beginning, proved to be a good bet in the long run.