# Parallel Tabu Search for Graph Coloring Problem

Jacek Dąbrowski[1] and Marek Kubale[2]

[1] Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics, Gdańsk, Poland, email: Jacek.Dabrowski@eti.pg.gda.pl
[2] Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics, Gdańsk, Poland, email: kubale@eti.pg.gda.pl

**Abstract.** *Tabu search* is a simple, yet powerful meta-heuristic based on local search that has been often used to solve combinatorial optimization problems like the *graph coloring problem*. This paper presents current taxonomy of *parallel tabu search* algorithms and compares three parallelization techniques applied to *Tabucol*, a sequential TS algorithm for graph coloring. The experimental results are based on graphs available from the DIMACS benchmark suite.

## 1 Introduction

Graph coloring has been an attractive field for more than a century with many thousands of papers dealing with the study of chromatic properties of graphs. As people were becoming used to applying the tools of graph theory to the solution of real-world technological and organizational problems, chromatic models appeared as a natural way of tackling many practical situations. One of the models of graph coloring with relatively many applications appears to be the vertex coloring problem, which we simply call the *graph coloring problem* (GCP).

A coloring of a graph $G = (V, E)$, where $V$ is the set of $n = |V|$ vertices and $E$ is the set of edges, is a mapping $c : V \mapsto 1..k$, such that for each edge $\{u, v\} \in E$ we have $c(u) \neq c(v)$. Optimization version of GCP is stated as follows: given a graph $G$, find a coloring with the minimum number $k$ of colors used. This number is referred to as $\chi(G)$, the *chromatic number of graph G*. The GCP is a well-known NP-hard combinatorial optimization problem.

*Tabu search*(TS) is a metaheuristic based on a local search approach: it is an iterative procedure that tries to improve on current solutions by exploring its *neighborhood* and the best candidate as the new current solution. The main concept in tabu search is a *tabu list* which keeps track of recently visited solutions to avoid cycling or getting trapped in a local optimum.

Parallel computation aims at solving problems quicker than sequential methods. In broad terms, this means either "find a solution of a similar quality faster" or "find a solution of a better quality in a comparable time". Parallel implementations are often more robust than the sequential ones, providing better solutions consistently over diverse sets of problem instances.

This paper is organized as follows. Section 2 reviews parallel tabu search models. Section 3 presents tabu search applications to GCP. Section 4 describes in detail three

parallel algorithms used in this work. Section 5 presents experimental results from several runs on standard benchmark graphs[1] as well as pseudorandom graphs.

## 2 Parallel tabu search

There are two main approaches to tabu search parallelization. It can be done at a *low level*, where the parallel processing is used only to speed up tasks with high computation cost (e.g., neighborhood evaluation). This means that the behavior of the search is the same as that of a sequential algorithm, the difference is in wall-clock time. *High level* parallelism means simultaneous operation of multiple searches either independent or cooperating.

In [3] Crainic, Toulouse and Gendreau introduced a three-dimensional taxonomy for parallel tabu search methods. The first dimension, *Search Control Cardinality*, decides whether the search process is controlled by a single *master* processor (*1-control, 1C*) or each of $p$ processors controls its own search (*p-control, pC*).

The second dimension, *Control and Communication Type*, is based on the communication scheme. There are four stages that define the operation mode (synchronous / asynchronous) and the type and amount of information shared. The first stage, *Rigid Synchronization* (*RS*), corresponds to simple synchronous communication with limited information exchange. For *1-control* approach *RS* means that communication is initiated exclusively by *master* and it is used only to delegate computing intensive tasks to *slaves*. For *p-control* every process performs its own search and there is no communication between them. The best solution is selected once all processes have stopped. The second stage, *Knowledge Synchronization*(*KS*), increases the level of communication, allowing knowledge exchange. For *1-control* the difference is in the complexity of the task assigned to *slaves*, for *p-control KS* means that all processes stop at a predetermined moment and begin a phase of information exchange. The information can be used to improve the individual searches.

Last two stages operate in asynchronous mode. It means that processes communicate after events (e.g., finding an improved solution) rather than at a specific stage of the algorithm or after predetermined number of moves. In the *Collegial* (*C*) stage each process executes a tabu search. When an improved solution is found, it is broadcast to other peers. The most advanced model is *Knowledge collegial* (*KC*). Here, the contents of communications are analyzed to retrieve additional information concerning global search trajectory and global characteristics of good solutions.

The third dimension, *Search Differentiation Strategy*, indicates whether the searches start from a single point or from different points in solution space and whether the used search strategies are the same or different. The four cases are: *SPSS: Single (initial)Point Single Strategy*, *SPDS: Single Point Different Strategies*, *MPSS: Multiple Points Single Strategy* and *MPDS: Multiple Points Different Strategies*.

## 3 Parallel tabu search algorithms for GCP

This section contains descriptions of three parallelization models used in this paper. All three are based on a well-studied *Tabucol* algorithm introduced in 1987 by Hertz and de

[1] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

Werra [5].

*Tabucol* is the first tabu search algorithm proposed for GCP. In particular it solves the decision version of the problem. For a given graph $G$ and $k$ the search space explored is the set of $k$-colorings of graph $G$. The goal is to find a coloring without any conflicting edges.

The evaluation function $f$ measures the number of conflicting edges in the solution. For a coloring $c : V \mapsto 1..k$ the value of $f(c)$ is equal to $|\{u, v\} \in E : c(u) = c(v)|$. *Tabucol* uses a simple *1-exchange* neighborhood, where a move is a pair $(v, i)$ denoting assignment of color $i$ to vertex $v$. After a move is performed the pair $(v, i)$ becomes a tabu move for $\lfloor L + \lambda f(c) \rfloor$ succeeding iterations. Most implementations use $\lambda = 0.6$ and choose $L$ randomly in [0..9].

Algorithms presented in this paper solve optimization version of the graph coloring problem. The initial coloring is generated using DSATUR greedy heuristic [2]. At the beginning of each iteration if the current solution $x$ is a valid coloring ($f(x) = 0$) the smallest color class is removed, and the vertices are reassigned. The tabu search is used to reduce the number of conflicting edges in the solution.

## 4 Implementation

The following subsections describe three models used in this work to solve GCP with parallel tabu search algorithm.

### 4.1 Master-slave search

The first algorithm is a low-level parallelization of *Tabucol*. The neighborhood $N$ is divided into $p$ subsets $N_i$. During each iteration every processor evaluates its part of the neighborhood and sends the best move to the *master*. The *master* evaluates the candidate moves, selects the best among them and broadcasts it. To speed up the exploration every process keeps its own copy of *tabu list*.

**Algorithm 1** Master-slave search

```
x ← DSATUR(G)                          ▷ use a greedy heuristic to generate initial solution
while stop-condition do
    while f(x) = 0 do                  ▷ while x is a valid coloring
        x ←REDUCE(x)                   ▷ reduce number of colors
    end while
    m ←best allowed move in N_i        ▷ ties are broken randomly
    if processor is slave then
        send m to master
        M ← receive selected move from master
    else
        moves[] ← receive moves from slaves
        M ← select best move from moves[]   ▷ ties are broken randomly
    end if
    x ← x + M                          ▷ perform the chosen move
    update tabu list
end while
```

According to the aforementioned taxonomy this algorithm should be classified as *1C/RS/SPSS*. The behavior of this algorithm is almost the same as that of a sequential implementation. The difference is that ties are broken at $N_i$ level, which means in fact that not all best moves have the same probability of being chosen. For large graphs the speedup is expected to be proportional to the number of processors.

### 4.2 Independent search

A completely different approach to parallelization has been used in the second algorithm. Here we use $p$ independent searches, each working on its own solution. Since there is no knowledge sharing between processors, the algorithm can be classified as *pC/RS*. As for the third dimension, *Search Differentiation Strategy*, two versions of the algorithm were compared - *MPSS* and *MPDS*. In the latter case the difference lies in the *tabu tenure*: $\lambda$ parameter is distributed evenly in the range $[0.4, 0.9]$.

This approach should broaden the search increasing the rate of success. A small improvement in speed is also expected due to random nature of the search process.

**Algorithm 2** Independent search

```
x ← DSATUR(G)                          ▷ use a greedy heuristic to generate initial solution
while stop-condition do
    if f(x) = 0 then
        while f(x) = 0 do              ▷ while x is a valid coloring
            x ←REDUCE(x)               ▷ reduce number of colors
        end while
        report the solution to master
    end if
    m ←best allowed move in N          ▷ ties are broken randomly
    x ← x + M                          ▷ perform the chosen move
    update tabu list
end while
```

### 4.3 Cooperating search

The last approach can be classified as *pC/KS/MPSS*. It is based on the independent search with the following improvement. Whenever a new valid coloring is found, it is sent to the *master*. The *master* reduces the number of colors and sends the new coloring to all processors. The *slaves* abandon their current search and use the received solution to start a new one. The tabu list is not copied.

## 5 Results

The experiments were performed on *holk* cluster in TASK Academic Computer Centre[2]. *Holk* has 256 1.3 GHz Intel Itanium 2 processors with 3MB L3 cache memory.

[2] http://www.task.gda.pl/

**Algorithm 3** Cooperating search

```
x ← DSATUR(G)                          ▷ use a greedy heuristic to generate initial solution
while stop-condition do
    if f(x) = 0 then
        while f(x) = 0 do                          ▷ while x is a valid coloring
            x ←REDUCE(x)                           ▷ reduce number of colors
        end while
        send x to master
    end if
    if processor is slave then
        if a new solution is available from master then
            x ← receive solution from master
        end if
    else
        if a better solution is available from any of the slaves then
            x ← receive solution from slave
            send x to all slaves
        end if
    end if
    m ←best allowed move in N                  ▷ ties are broken randomly
    x ← x + M                                  ▷ perform the chosen move
    update tabu list
end while
```

Parallel implementation was based on LAM-MPI. *MPI*, Message Passing Interface, is a library of routines that can be called from Fortran, C, C++ and Ada programs[3]. *MPI*'s advantage over older message passing libraries is that it is both portable (because *MPI* has been implemented for almost every distributed memory architecture) and fast (because each implementation is optimized for the hardware it runs on).

### 5.1 Results for master-slave search

The first experiment was to determine how low-level parallelization influences the speed of the search. A large random graph: `DSJC1000.5` has been colored using 1, 2, 4 and 8 processors. All runs had a time limit of one hour. Table 1 presents results of those runs: number of tabu search iterations calculated within time limit, number of iterations calculated per second and *speedup*: the ratio of the time of sequential execution to the time of parallel execution.

It was observed that for the *Tabucol*-based algorithm the parallel implementation performs worse than the sequential one not only in terms of total computation time, but also in terms of wall-clock time. Although the neighborhood evaluation is a task with the highest computation cost ($O(|V| * k)$), it is not large enough to be a reason for parallelization. Therefore this approach was not investigated further.

[3] http://www.mpi-forum.org/

**Table 1.** Results for master-slave search

| number of processors: | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| number of iterations: | 4015000 | 3734000 | 3661000 | 2861000 |
| iterations per second: | 1115 | 1037 | 1016 | 795 |
| speedup | 1 | 0.93 | 0.912 | 0.712 |

### 5.2 Results for independent search

In this experiment the behavior of independent search threads was investigated. The best solution of all threads is considered to be the best solution of the parallel algorithm. Therefore the average performance of the parallel search is better than the performance of a single search. Table 2 shows how long it takes to find a valid $k$-coloring for different values of $k$.

**Table 2.** Results for independent search

| | | 1 processor | | 10 processors | |
|---|---|---|---|---|---|
| graph | $k$ | successful runs | time [s] | successful runs | time [s] |
| `DSJC1000.5` | 115 | 10/10 | 3 | 3/3 | 0 |
| $\|V\| = 1000$ | 105 | 10/10 | 42 | 3/3 | 29 |
| $\|E\| = 249826$ | 100 | 10/10 | 142 | 3/3 | 104 |
| | 95 | 7/10 | 1381 | 3/3 | 887 |
| | 94 | 2/10 | 2503 | 3/3 | 2427 |
| `DSJC500.5` | 60 | 10/10 | 4 | 3/3 | 0 |
| $\|V\| = 500$ | 55 | 10/10 | 32 | 3/3 | 24 |
| $\|E\| = 6262$ | 54 | 8/10 | 66 | 3/3 | 44 |
| | 53 | 5/10 | 421 | 3/3 | 221 |

### 5.3 Single strategy vs different strategies

Some experiments were performed to determine if using independent searches with different values of $\lambda$ and $L$ parameters defining the *tabu tenure* would improve the behavior of the algorithm. For $p$ processors $\lambda$ is distributed evenly in $[0.4, 0.9]$ range and $L$ in $[5..20]$. No significant differences were observed when compared to the *MPSS* model.

The strategies might be differentiated by other characteristics, e.g. neighborhood structure. These possibilities have not been investigated in this work.

### 5.4 Independent search vs cooperating search

More experiments were performed to check if knowledge sharing would improve the results. Table 3 provides results for several graphs from the DIMACS suite.

For the two large graphs: `DSJC1000.5` and `flat1000_50_0` the time limit was set to two hours, for smaller graphs it was one hour. The runs were conducted using 10

processors.

Knowledge sharing improves the behavior of the algorithm. For easy colorings that are found within one minute the speedup is not obvious because of the communication overhead created by frequent colorings exchange. Cooperating search also improves the success rate within the specified time limit.

**Table 3.** Results for cooperating search

| graph | $k$ | *pC/RS/MPSS* successful runs | *pC/RS/MPSS* time [s] | *pC/KS/MPSS* successful runs | *pC/KS/MPSS* time [s] |
|---|---|---|---|---|---|
| `DSJC1000.5` | 110 | 3/3 | 14 | 3/3 | 10 |
| $\|V\| = 1000$ | 100 | 3/3 | 142 | 3/3 | 116 |
| $\|E\| = 249826$ | 95 | 3/3 | 887 | 3/3 | 754 |
| | 94 | 3/3 | 2427 | 3/3 | 1908 |
| | 93 | 0/3 | - | 1/3 | 3275 |
| `DSJC500.5` | 55 | 3/3 | 24 | 3/3 | 26 |
| $\|V\| = 500$ | 54 | 3/3 | 44 | 3/3 | 38 |
| $\|E\| = 62624$ | 53 | 3/3 | 221 | 3/3 | 240 |
| `flat1000_50_0` | 110 | 3/3 | 6 | 3/3 | 7 |
| $\|V\| = 1000$ | 100 | 3/3 | 184 | 3/3 | 116 |
| $\|E\| = 245000$ | 95 | 3/3 | 311 | 3/3 | 346 |
| $\chi = 50$ | 93 | 3/3 | 3816 | 3/3 | 1267 |
| | 92 | 1/3 | 5806 | 3/3 | 6748 |
| `flat300_28_0` | 36 | 3/3 | 3 | 3/3 | 3 |
| $\|V\| = 300$ | 35 | 3/3 | 8 | 3/3 | 5 |
| $\|E\| = 21695$, $\chi = 28$ | 34 | 1/3 | 67 | 3/3 | 63 |
| `le450_25c` | 28 | 3/3 | 3 | 3/3 | 3 |
| $\|V\| = 450$ | 27 | 3/3 | 56 | 3/3 | 64 |
| $\|E\| = 17343$, $\chi = 25$ | 26 | 0/3 | - | 0/3 | - |

### 5.5 Results for random graphs

This section gives the results of experiments on random graphs. A random graph $G_{n,p}$ has $n$ vertices and every two vertices are joined by an edge with probability $p$. For every pair of $n$ and $p$ five graph instances were generated.

In [6] Johri and Matula have presented means to calculate $\chi*$, the estimated chromatic number and $\chi_*$, a probabilistic lower bound for the chromatic number. In our experiments the probability that $\chi(G_{n,p}) \geq \chi_*$ was greater than $1 - 10^{-6}$.

Tables 4 and 5.5 compare greedy algorithm DSATUR and sequential tabu search with cooperating parallel tabu search using 4 and 16 processors. The results were averaged over five runs for each of the five instances of $G_{n,p}$. Computation time of DSATUR algorithm was less than one second for all graphs.

In the first experiment tabu search was allowed to run for a specified amount of time. Table 4 shows the average number of colors used in the best coloring found.

Based on the results from the first experiment tabu search was configured to look for solutions using at most $k$ colors. Table 5.5 shows the success rate and the average time needed to find a solution of a specified quality for $p = 0.5$. The run was considered unsuccessful if a solution was not found within the time limit of sixty seconds for every hundred vertices.

Using 16 processors improved the best coloring found for each instance by one or two colors. Cooperating search found good solutions faster and with greater probability.

**Table 4.** Average number of colors used

| $n$ | $p$ | $\chi*$ | $\chi_*$ | DSAT | time limit [s] | processors 1 | 4 | 16 |
|---|---|---|---|---|---|---|---|---|
| 100 | 0.25 | 10 | 6 | 10.3 | 30 | 9.0 | 9.0 | 9.0 |
| | 0.50 | 16 | 11 | 17.8 | | 16.0 | 15.6 | 15.1 |
| | 0.75 | 26 | 19 | 30.0 | | 25.9 | 25.2 | 25.0 |
| 300 | 0.25 | 19 | 14 | 22.5 | 90 | 19.0 | 18.5 | 18.4 |
| | 0.50 | 35 | 27 | 43.2 | | 35.7 | 35.4 | 35.1 |
| | 0.75 | 57 | 48 | 72.7 | | 60.6 | 59.6 | 58.9 |
| 500 | 0.25 | 27 | 20 | 33.1 | 150 | 27.5 | 27.3 | 27.0 |
| | 0.50 | 50 | 41 | 65.2 | | 54.1 | 53.6 | 53.1 |
| | 0.75 | 86 | 74 | 110.6 | | 92.4 | 91.6 | 91.0 |

**Table 5.** Average times of finding a valid $k$-coloring

| $n$ | $\chi*$ | DSAT | $k$ | 1 processor succ | time($\sigma$) | 4 processors succ | time($\sigma$) | 16 processors succ | time($\sigma$) |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 16 | 17.7 | 16 | 25 | 0.1(0.3) | 25 | 0.2(0.6) | 25 | 0.0(0.2) |
| | | | 15 | 3 | 2.3(3.3) | 11 | 0.7(1.1) | 22 | 2.4(9.8) |
| 200 | 26 | 31.0 | 26 | 24 | 7.7(10.0) | 25 | 2.8(4.0) | 25 | 1.6(1.4) |
| | | | 25 | 1 | 35.0(0.0) | 3 | 8.0(4.3) | 16 | 28.9(30.5) |
| 300 | 35 | 42.6 | 36 | 23 | 19.7(22.4) | 25 | 6.0(3.0) | 25 | 3.4(1.6) |
| | | | 35 | 6 | 38.8(25.5) | 19 | 37.2(33.4) | 25 | 23.2(19.6) |
| 400 | 43 | 54.5 | 45 | 25 | 40.9(30.9) | 25 | 22.1(16.9) | 25 | 13.3(5.9) |
| | | | 44 | 6 | 64.8(36.3) | 16 | 85.2(45.8) | 22 | 42.8(29.2) |
| 500 | 50 | 64.8 | 54 | 25 | 78.4(47.4) | 25 | 46.6(17.0) | 25 | 31.0(8.7) |
| | | | 53 | 11 | 122.5(40.3) | 19 | 164.7(69.2) | 25 | 105.9(61.7) |

## 6 Concluding remarks

For the coarse-grained architecture of *holk* cluster it was shown that the simple master-slave algorithm is slower than the sequential algorithm executed on a single processor. However this might not be true for a shared-memory architecture environment. The

multiple path strategy proved to be successful in improving the quality of best solution found. It also reduced time needed to find $k$-colorings for given values of $k$.

Many recently proposed graph coloring techniques use tabu search or other local search algorithms as means to improve solutions. Galinier and Hao [4] in their genetic-local search hybrid method perform a TS before inserting the result of a crossover into the population. In *variable neighborhood search* [1] during every iteration the algorithm makes a big, variable move (change) to current solution and tries to improve it with *Tabucol*. Both techniques could perform significantly better if they were to make use of parallel computation. This possibility should be investigated further.

## Bibliography


[1] C. Avanthay, A. Hertz, and N. Zufferey. Variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151:379–388, 2003.

[2] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[3] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search heuristics. *INFORMS Journal of Computing*, 9:61–72, 1997.

[4] P. Galinier and J.-K. Hao. Hybrid evolutionary algorithm for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[5] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[6] A. Johri and D.W. Matula. Probabilistic bounds and heuristic algorithms for coloring large random graphs. Technical report, Southern Methodist University, 1982.