Lecture October 8

The results of the Contest Task 2 were presented and discussed. The winner applied an iterated greedy algorithm that at each iteration uses the greedy heuristic to color a new permutation of vertices on the basis of the color classes attained in the previous iteration. The second qualified uses a combination of local search and random redistribution of vertices in a limited number of color classes. The script in R for producing the output presented at the lectures is available at \texttt{analysis.Task2.R}.

Dynamic local search and iterated local search were then treated and evolutionary algorithms were introduced. As exercise, we saw different ways to apply Simulated Annealing and Tabu Search to the GCP.

In the next lecture we will continue treating evolutionary computation and swarm intelligence for optimization.

The Task 3 of the implementation contest has been posed.

Bibliographical Notes

- Dynamic local search and iterated local search are described on pages 82-89 of the text book.
- Evolutionary Algorithms are described on pages 100-105 of the text book.

There are many examples of applications of the metaheuristics seen so far sparse throughout the book. This is a list of recommended readings:

- SAT problem
  - GSAT (randomized iterative improvement + tabu search) pages 267–272
  - WalkSAT (randomized iterative improvement + tabu search) pages 274–276

- CSP
  - Tabu search pages 305

- MAX-SAT problem
  - Dynamic local search pages 324–329
  - Tabu Search pages 329–331
  - Iterated local search pages 331–335

- TSP
  - Iterated local search pages 384–399
  - Evolutionary algorithm pages 399-405
Implementation Contest - Task 3

The third Task of the Implementation Contest asks to submit a peak performance algorithm for the approximation of the chromatic number of a graph within a given time limit.

The time limit for each single run is 360 seconds for C/C++ users and 420 seconds for Java users. Times refer to the machine woglindle.imada.sdu.dk of the terminal room.

The algorithms will be evaluated on the flat instances and some new instances (the whole new set is available for download Instances-Task-3.tgz). The instances in the set are generated in order to admit different kind of colorings, ranging from equi-partite classes to highly variable classes. The algorithms will be allowed to run for the same time and the comparison will be done on the basis of solution quality only. If a participant designs more than one algorithm he/she must choose the one deemed the best and submit only that one.

The suggestion is to use the methods developed until now and enhance them with new methods. A list of possible methods to try is given in the section below. There is, however, freedom to choose methods and hybridizations thereof not listed below.

The same submission details and input output formats remain those in Task 1. Make sure your program receives a flag -t SECONDS as an input for the time limit. In addition, make sure that in the output file there will be the log of best solution found with this syntax:

```
best 40 time 100
```

where the value after `best` represents the best proper coloring found and the value after `time` the time to find this solution. In the standard output, instead, as usual, there should be only the best overall and the exit time.

The deadline is at 12.00 of Wednesday 24th October.

A list of possible methods for GCP

This is a list of algorithms for graph coloring that can be found in the literature and for which successful results were reported.

**Strategy 1: k fixed, complete colorings**

**Tabu search with 1-exchange neighborhood.** Tabu search algorithms based on the 1-exchange neighborhood choose at each iteration a best non-tabu or tabu but aspired neighboring candidate solution from the restricted 1-exchange neighborhood. If a 1-exchange move puts vertex $v$ from color class $C_i$ into $C_j$, it is forbidden to re-assign vertex $v$ to $C_i$ in the next $t_t$ steps; the tabu status of a neighboring solution is overruled if it improves over the best candidate solution found so far (aspiration criterion). If more than one move produces the same effect on the evaluation function, one of those moves is selected uniformly at random. The tabu list length is set to $t_t = \text{random}(A) + \delta \cdot |V^C|$, where $\text{random}(A)$ is an integer uniformly chosen from $\{0, \ldots, A\}$ and $\delta$ and $A$ are parameters. Since $t_t$ depends on $|V^C|$, the tabu list length varies dynamically with the evaluation function value.

**Min-Conflicts heuristic.** It uses the same two-stage selection process as the min-conflicts heuristic, but in the second stage it only allows to move the vertex into a color class that is not tabu. If all color classes are tabu, one is chosen randomly. One advantage of this neighborhood examination strategy is that it does not require the usage of sophisticated caching and updating schemes; hence it allows for an easier implementation. In addition, the chances of cycling are reduced due to the random choices especially in the first stage of the selection process, allowing for shorter tabu lists.
Guided local search. Guided local search (GLS) uses an augmented evaluation function \( g' \) defined as

\[
g'(C) = g(C) + \lambda \cdot \sum_{i=1}^{\mid E \mid} w_i \cdot I_i(C),
\]

where \( g(C) \) is the usual evaluation function, \( \lambda \) a parameter that determines the influence of the penalties on the augmented cost function, \( w_i \) the penalty weight associated to edge \( i \), and \( I_i(C) \) an indicator function that takes the value 1 if the end points of edge \( i \) are in conflict in \( C \) and 0 otherwise. The penalties are initialized to 0 and are updated each time an iterative improvement algorithm reaches a local optimum of \( g' \). The modification of the penalty weights is done by first computing a utility \( u_i \) for each violated edge, \( u_i = I_i(C)/(1 + w_i) \), and then incrementing the penalties of all edges with maximal utility by one. The underlying local search is a best-improvement algorithm in the restricted 1-exchange neighborhood. Once a local optimum is reached, the search continues for a maximum number of \( sw \) plateau moves before the evaluation function \( g' \) is updated.

Iterated local search. Tabu Search described above is run until the best solution found does not change for \( l_{LS} \) iterations. A perturbation is then applied to the best coloring found so far and Tabu Search is run again. In the perturbation, a number \( k_r, k_r < k \), of color classes is randomly chosen and the color class membership of all vertices in those color classes is changed. The ROS heuristic bounded by \( k \) and with the further strong constraint of avoiding the re-insertion of a vertex into its previous color class is used to accomplish this task. The tabu list of Tabu Search is emptied before applying the perturbation, while the exchanges caused by the perturbation are inserted in the tabu list.

Hybrid iterated greedy and tabu search. This hybrid algorithm first generates an initial solution with one of the construction heuristics. Then performs some steps of iterated greedy, that is, vertices are sorted using the current coloring in some order of the independent sets and a new coloring is formed using the greedy heuristic. Then a tabu search phase is applied. This starts from an iteration of iterated greedy and uses the first \( k' = k - c \) colors for the greedy reconstruction. Then vertices that remain uncolored (those from the \( c \) exceeding sets) are distributed in the independent sets that creates fewest conflicts. Tabu search as described above is applied to make the coloring feasible. If after a prescribed number of iterations Tabu search fails to find a feasible coloring then the number of colors is increased by one. This means that a newest partition element is initially empty but vertices in conflict can move to it. The overall algorithm oscillates between an iterated greedy phase and a tabu search phase.

Evolutionary algorithms. Evolutionary algorithm for the GCP starts with a population \( P \) of candidate solutions, which is initialized by using the ROS construction heuristic restricted to \( k \) colors (with \( k \) determined by the DSATUR heuristic), and then iteratively generates new candidate solutions by first re-combining two members of the current population that are improved by local search. For the recombination, the greedy partition crossover (GPX) is used. Starting with two candidate partitionings (parents) \( C^1 = \{C^{1}_1, \ldots, C^{1}_k\} \) and \( C^2 = \{C^{2}_1, \ldots, C^{2}_k\} \), GPX generates a candidate solution (offspring) by alternately selecting color classes of each parent. At step \( i \) of the crossover operator, \( i = 1, \ldots, k \), GPX chooses a color class with maximal cardinality from parent \( C^1 \) (if \( i \) is odd) or from parent \( C^2 \) (if \( i \) is even). This color class will become color class \( C_i \) of the offspring. Once \( C_i \) is chosen, the vertices that belong to it are removed from both parents. The vertices that remain in \( C^1 \) and \( C^2 \) after step \( k \) are added to a color class of the child, which for each vertex is chosen uniformly at random. The new candidate partitioning returned by GPX is then improved by Tabu Search, run for \( l_{LS} \) iterations, and it is inserted in the population \( P \) replacing the worse parent. The population is re-initialized if the average distance between colorings in the population falls below a threshold of 20.

Other methods. A greedy randomized adaptive search procedure uses a randomization of RLF for the candidate solution construction and an iterative improvement algorithm in the 1-exchange neighborhood. Ant colony optimization (ACO) can also been applied to the graph coloring problem.

Strategy 2: \( k \) fixed, partial colorings

Distributed coloration neighborhood search. The distributed coloration neighborhood algorithm can be seen as an ILS algorithm that uses a simulated annealing (SA) algorithm for the local search. The SA algorithm is based on the \( i \)-swap neighborhood and is run for \( I \) iterations or until a certain solution quality threshold is passed. A neighbor of \( C \) in the \( i \)-swap neighborhood is obtained by moving a vertex \( v \) from \( C_{imp} \) (the impasse class, that is, the class of vertices not yet colored) into another color class \( C_i \), followed by moving all vertices

\[
3
\]
of $C_i$ that share an edge with $v$ into $C_{imp}$ so that the partial coloring $\bar{C}$ remains feasible. Upon termination of the SA algorithm, a perturbation is applied that is defined by a random $s$-chain exchange that moves from the current coloring configuration to a new one with the same solution quality. An $s$-chain exchange can be seen as a generalization of Kempe chain exchange. It is defined through a vertex $v$ and an ordered sequence of non-empty color classes $C_1, \ldots, C_s$, where $v \in C_1$, all color classes are distinct, and $s \leq k$. From this sequence, a directed graph with vertex set $V' = C_1 \cup \ldots \cup C_s$ and arc set $A = \{(u, w) \mid (u, w) \in E, u \in C_i$ and $w \in C_{i+1}\}$ is derived (for convention, $C_{s+1} = C_1$). In an $s$-chain, each vertex that is reachable from $v$ in the digraph $(V', A)$ is moved from color class $C_i$ to $C_{i+1}$. Note that an $s$-chain, where all vertices in $V'$ are reachable, would simply correspond to a relabelling of the color classes and, hence, would result in the very same partitioning; therefore, such a total $s$-chain is to be avoided and the neighborhood is restricted to non-total $s$-chains.

**Tabu Search.** A tabu search algorithm based on the $i$-swap neighborhood was also applied. The tabu criterion in this algorithm forbids adding into $C_i$ vertices adjacent to a vertex $v$ that was moved into a color class $C_i$. This interdiction acts for the $tt$ iterations successive to the move of $v$.

**Strategy 3: $k$ variable, complete colorings**

**Simulated annealing with Kempe chain neighborhood.** A version of Simulated annealing allows only feasible colorings and uses the Kempe chain neighborhood, while another version allows infeasible colorings and uses the 1-exchange neighborhood. SA uses the evaluation function $g(C) = -\sum_{i=1}^{k} |C_i|^2 + \sum_{i=1}^{k} 2|C_i||E_i|$ and starts from an initial partitioning generated by the ROS heuristic. At each iteration of SA with Kempe Chains, a neighboring solution is generated in three steps; firstly, a non-empty color class $C_i$, a vertex $v \in C_i$, and a non-empty color class $C_j$ are chosen uniformly at random but avoiding that $C_i$ and $C_j$ form a full connected component, which would result simply in a re-labelling of the color classes; secondly, the Kempe chain $K_{ij}$ of color classes $C_i$ and $C_j$ that contains vertex $v$ is determined; thirdly, the Kempe chain exchange is applied. The generated neighboring candidate solution $C'$ is always accepted if it improves over the current candidate solution $C$ and otherwise it is accepted with a probability of $\exp((g(C) - g(C'))/T)$, where $T$ is the temperature modified according to a rather standard cooling schedule.