2nd Lt David Crow

ENG/20M

CSCE 686 Advanced Algorithms, Homework 4a

­

**Problem 1 – Talbi 1.4**

The size of a problem’s input is not the only factor that describes the difficulty of the problem; typically, the problem’s structure is more important. For a given problem, it’s possible that small instances cannot be solved to optimality, but that large instances can be solved exactly. Show for some classical optimization problem (e.g. satisfiability, knapsack, bin packing, vehicle routing, and set covering) that some small instances can’t be solved exactly, while some large instances are solved to optimality by state-of-the-art exact optimization methods.

In general, as the size of an NP problem increases, the difficulty of said problem increases as well. However, there exists a *phase transition* in some NP problems. After this phase transition, the problem’s difficulty actually tends to *decrease* as the problem’s size increases [1]. This is an interesting phenomenon, and we explore it more here as it relates to the maximal independent set (MIS) problem.

In a graph , an *independent set* is a set of vertices such that and, additionally, . In English, then, an independent set consists of vertices of such that no vertex in is connected to any other vertex in . A *maximal independent set* is an independent set such that no vertex can be added to such that is still an independent set [2].

The MIS problem is an NP-hard optimization problem [2]. Additionally, it is an NP-hard problem with a phase transition [3]. As shown by Barbosa and Ferreira, for randomly-generated graphs of increasing size, the independence number – that is, the size of the maximal independent set – decreases. For this reason, it becomes easier to find the independence number of graphs for which the size is beyond the phase transition. In other words, it becomes easier to compute the independence number, and thus solve the MIS problem exactly, for remarkably large graphs because the number of disjoint vertices naturally decreases as graph size increases [3].

We see, then, that the number of simple problem instances grows faster than the number of complex problem instances as graph size increases. Eventually (after the phase transition, that is), the number of simple instances dominates in the set of all instances for a given size, and it thus becomes easier to solve MIS on average.

References

1. <https://www.ijcai.org/Proceedings/91-1/Papers/052.pdf>
2. <https://en.wikipedia.org/wiki/Independent_set_(graph_theory)>
3. <https://arxiv.org/pdf/cond-mat/0309518.pdf>

**Problem 2**

Using the [Bron-Kerbosch (BK) Wikipedia page](https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm), add the standard search elements as comments to the BK pseudocode (without pivoting). Discuss the symbols in this BK pseudocode as they relate to the symbols in Christofides’s BK MIS algorithm development process.

As given by Wikipedia, the BK pseudocode (without pivoting) is as follows:

1 BronKerbosch1(, , ):

2 if and are both empty:

3 report as a maximal clique

4 for each vertex :

5 BronKerbosch1(, , )

6 :=

7 :=

According to a document handed out during class, the standard search elements are as follows:

* Set of candidates – a set of possible next-state candidates (implicit, explicit)
* Next state generator – generates the next state candidates from a parent state candidate
* Feasibility – determines if various next-state candidates meet the feasibility criteria or constraint
* Selection – a function to select/extract/delete one or more of the feasible next-state candidates as most promising
* Solution – a function to determine if the current-state candidate is an acceptable solution; if not, eliminate it
* Objective – a function that reflects the selected optimization criteria
* Heuristics – a function that, as appropriate, contains algorithmic strategies usually based upon a reduced set of candidates via insight from problem domain structure and objective function

We can see that these search elements are present in the pseudocode. Specifically,

* Set of candidates
  + Although such a set is not explicitly defined, we can generate our set of candidates from , , and , which are passed into in line 1. Every possible combination of the vertices in , when added to the vertices in , is a candidate.
* Next state generator
  + The recursive call on line 5 generates the next state. Specifically, we update our current state to find our next state(s) . We say state(s) because it’s certainly possible that , so there are multiple vertices with which we can generate a new state.
* Feasibility
  + In line 5, we see that concerns the feasibility of the next state. Our clique is not maximal if there exists at least one node such that has an edge to every other vertex in . The function – when intersected with and with – determines whether gives us a feasible solution (that is, if and are empty, then adding to can’t lead to a maximal clique). Of course, we don’t know the result of this feasibility check until the recursive call, so line 2 is also part of our feasibility check.
* Selection
  + In line 6, we delete those candidates (and all further candidates in the search tree) that include .
* Solution
  + Line 2 checks whether the current is a valid solution. We know that, if

, then is a maximal clique (because there are no remaining vertices with which we can expand our clique).

* Objective
  + In line 2, we don’t simply check whether the candidate is a solution; we check whether the candidate is an *optimal* solution. Because BK needs to return all maximal cliques in our graph, returning all cliques – whether or not they are maximal – does not meet our objective. In line 2, we ensure we only return those cliques that are already maximal. In other words, if it’s possible that another vertex can be added to without breaking the maximal constraint, then line 2 will ensure that we don’t yet return the clique. In this way, we ensure we meet our objective.
* Heuristics
  + In line 5, we consider only those vertices in and that are neighbors of . In doing we, we limit the search space.
  + In lines 6 and 7, we move from to , which ensures that we stop considering for our maximal cliques. This also limits the search space.