algorithms

MDPI

Article

The Use of an Exact Algorithm within a Tabu Search Maximum Clique Algorithm

Derek H. Smith [1,*], Roberto Montemanni [2] and Stephanie Perkins [1]

[1] School of Computing and Mathematics, University of South Wales, Pontypridd, CF37 1DL Cardiff, UK; stephanie.perkins@southwales.ac.uk

[2] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Via Amendola 2 (Pad. Morselli), 42122 Modena MO, Italy; roberto.montemanni@unimore.it

* Correspondence: derek.smith@southwales.ac.uk

Received: 25 August 2020; Accepted: 22 September 2020; Published: 4 October 2020

Abstract: Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A clique C of G is a subset of the vertices of V with every pair of vertices of C adjacent. A maximum clique is a clique with the maximum number of vertices. A tabu search algorithm for the maximum clique problem that uses an exact algorithm on subproblems is presented. The exact algorithm uses a graph coloring upper bound for pruning, and the best such algorithm to use in this context is considered. The final tabu search algorithm successfully finds the optimal or best known solution for all standard benchmarks considered. It is compared with a state-of-the-art algorithm that does not use exact search. It is slower to find the known optimal solution for most instances but is faster for five instances and finds a larger clique for two instances.

Keywords: combinatorial optimization; maximum clique; hybrid algorithm; tabu search; benchmarks

1. Introduction

Exact algorithms for the maximum clique problem are now remarkably efficient, but, for larger problems, heuristic algorithms are necessary. Tabu search is an effective metaheuristic for the maximum clique problem. This paper evaluates the option of including an exact algorithm within the tabu search, and considers the best exact algorithm to use. The candidate exact algorithms all use a graph coloring upper bound. This approach was used in the specific context of permutation code constructions in [1]. The final tabu search algorithm described, modified from a published algorithm [2], finds best known solutions to standard instances but is normally slower than a state-of-the-art tabu search algorithm [3] to find these solutions. However, it is more competitive for harder instances and actually finds significantly larger cliques for two open instances derived from the construction of permutation codes.

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A clique C of G is a subset of the vertices of V with every pair of vertices of C adjacent (a complete subgraph). A maximum clique is a clique with the maximum number of vertices. Problems involving cliques arise in many applications surveyed in [4], including bioinformatics, examination planning, location problems, signal transmission analysis, and social network analysis. Applications of the maximum clique problem in radio frequency assignment are described in [5]. The problem also arises in the construction of various types of error-correcting code with the maximum number of codewords [6,7]. Note that the problem of finding a maximum clique in the graph G is equivalent to the problem of finding a maximum independent set in the complement graph $\overline{G}$.

The decision form of the clique problem (does there exist a clique of size k) was proved to be one of the original 21 NP-complete problems in a classic paper [8]. It follows that the maximum clique problem is NP-hard. When exact methods are impractical, heuristic methods are used.

Algorithms for the maximum clique problem, both exact and heuristic, are surveyed in [4]. The algorithm Swap-Based Tabu Search (SBTS) presented in [3] is the only heuristic algorithm compared in [4] that is able to find the optimal or best known solution over a wide range of instances. A more recent algorithm HTS (Hybrid Tabu Search) [2] also finds the optimal or best known solution over these instances and adds some further instances based on the construction of permutation codes [7]. HTS makes use of both an exact and a pseudoexact inner solver. In the tabu search in HTS, a number of carefully selected vertices are removed from the current clique before an exact or pseudoexact algorithm is applied to the subgraph induced by the set of vertices of G adjacent to all vertices in the reduced clique.

As well as tabu search and other local search methods, there are also mathematical programming approaches, together with other methods, such as genetic algorithms, ant colony optimization, chemical reaction optimization, and particle swarm optimization. Hybrid and parallel versions of these algorithms also exist. The reader is referred to the survey [4] for an extensive list of references. Some later references to the maximum clique and related problems can be found in [9–13]. When comparable, these later approaches do not always match SBTS or HTS. For this reason, the current paper concentrates on tabu search. Although there are many effective algorithms, there is always scope for improvement on large and hard instances beyond the standard benchmarks.

There are four objectives of this paper. The first is to determine the best exact algorithm to replace the exact and pseudoexact algorithms used in HTS. This allows a comparison of the resulting tabu search algorithm, incorporating the best exact algorithm found, with a more standard tabu search approach. Initially, a comparison of various exact algorithms applied to benchmarks used in the comparison in [4] is carried out. The best candidates are then incorporated into the HTS algorithm and further compared, leading to a somewhat different rank ordering of methods. The second objective is to consider and implement a small number of other improvements to HTS. The third is to dramatically simplify the number of parameters in HTS by finding methods to determine most automatically. Finally, the resulting new algorithm HTS2 (Code for HTS2 is available in the Supplementary Material) can be compared with the original HTS and SBTS on the same machine. The overall aim of the work is to determine whether a tabu search algorithm including exact search can match SBTS on standard instances and find larger cliques than SBTS for some hard instances. This requires consideration of some new hard instances beyond those used in [3]. The new instances here are motivated by the authors' interest in the construction of permution codes.

2. Exact Algorithms with a Coloring Bound

Exact algorithms for the maximum clique problem are surveyed in [4]. The basis for many improved algorithms is the algorithm of Carraghan and Pardalos [14], and this is also the basis for the exact and pseudoexact algorithms used in HTS [2]. Within HTS,this algorithm works with a current clique F and a potential expansion set $S = N(F)$ of vertices adjacent to all vertices of F. Vertices of S are selected in turn and removed from S. The new potential expansion set S' is computed and the selected vertex is added to F to create a set F'. If the potential expansion set is empty, and a new best clique is obtained, the clique is assigned to a global variable *Best*. Otherwise, subject to a pruning condition, the algorithm is applied recursively to the new potential expansion set S'. The usual condition is that pruning of the search tree takes place unless $|F'| + |S'|$ is greater than the size of the best clique found so far in the exact algorithm. Within HTS, extra pruning is used involving an external lower bound arising from the best clique found so far in the metaheuristic. Pseudoexact search uses a probable upper bound for pruning derived from an evaluation of typical subproblems solved for the instance within HTS.

A major improvement arises from the use of a vertex coloring bound.

Definition 1. *A vertex coloring of a graph G is an assignment of colors to the vertices of G such that adjacent vertices are assigned different colors.*

Definition 2. *The chromatic number of a graph G is the minimum number of colors in a vertex coloring.*

The following proposition is well known:

Proposition 1. *The chromatic number of a graph G is an upper bound for the number of vertices in a maximum clique.*

Proof. Every vertex of the clique must be assigned a different color. □

The colors can be represented as positive integers $1, 2, \ldots$. Thus, any upper bound k for the chromatic number of the subgraph induced by S' (obtained from a coloring of S' with k colors) can replace $|S'|$ in the pruning to reduce the size of the search, sometimes dramatically. Then, options to be considered here are i) the best coloring algorithm to use, which must be very fast as the exact algorithm is used many times in hybrid tabu search, ii) whether to locally color each subgraph induced by S', which may be expensive, or to color the initial subproblem and use the colors inherited by the subgraph induced by S', or a combination of the two, iii) whether to re-order the vertices of S' in non-increasing order of color within the exact solver.

2.1. Initial Evaluation of Exact Algorithms with a Coloring Bound

Improvements to the Carraghan and Pardalos algorithm (denoted as "C&P") using a coloring bound are of three basic types. The first, denoted as "*local coloring*", applies a coloring algorithm to the original subproblem and to each new set S' created as the depth of the search tree is increased. Clearly the coloring algorithm has to be very fast. The second, denoted as "*start coloring*", applies the coloring algorithm to the original subproblem, and the color assigned to each vertex is inherited by that vertex in the set S'. The number of colors actually used in S' can then be computed to give a (usually weaker) upper bound. The third, denoted as "*start and local coloring*", is a hybrid. Start coloring and inheriting takes place as in "*start coloring*", but, if the upper bound obtained does not lead to pruning, then local coloring also takes place. This may give a smaller lower bound that does lead to pruning.

The coloring algorithm used is a simple greedy algorithm. The vertices of the set are colored in turn, and each vertex is colored with the smallest available color. Two variations of this are applied to start coloring only. The *saturation degree* of a vertex is obtained as follows. Once a new vertex has been colored, the saturation degree of an uncolored vertex is the number of colors in its neighborhood. Vertices are then colored in non-increasing order of saturation degree. This is denoted as "*saturation start*". The second very similar variation applied to start vertices uses Degree of Saturation (DSatur) coloring [15], itself based on saturation degree, but using a degree ordering to select the first vertex and to break ties. This is denoted as "*DSatur start*".

Another option that can be used is to order the vertices of each new set S' (created as the depth is increased) in non-increasing order of color. This means that, as subsequent vertices of S' are considered, the coloring upper bound tends to decrease quickly and proves particularly effective. This option is denoted by "*ordered*" in the following tables.

There is an important difference between the evaluation here and the general evaluation in [4]. Within HTS,many of the calls to the exact or pseudoexact algorithm supply a good lower bound. Thus, the evaluation here is carried out with the best available lower bound supplied. The time in seconds to complete the algorithms for a number of DIMACS instances [16] used in [4] are shown in Tables 1 and 2. No single algorithm is the best for all instances, but clearly vertex ordering is particularly helpful.

Table 1. Comparison of run times (in seconds) for four exact algorithms for the maximum clique problem.

Instance	C&P	C&P Local Coloring	C&P Start Coloring	C&P Start and Local Coloring
brock400_2	30,904	6407	30,538	5842
brock400_4	15,260	1420	11,216	1332
brock800_2	191,613	74,696	183,831	73,537
brock800_4	127,564	37,835	113,604	37,599
keller5	unterminated	unterminated	unterminated	unterminated
p_hat300-3	25,312	1110	11,884	960
p_hat700-2	unterminated	22,099	unterminated	20,523
p_hat1500-1	12	17	15	17

Table 2. Comparison of run times (in seconds) for four improved exact algorithms for the maximum clique problem using ordering. The algorithms in the second and third columns are variations of those in columns 3 and 5 of Table 1 using ordering. The fourth and fifth columns are variations of the third column using saturation or DSatur (Degree of Saturation) start, respectively.

Instance	C&P with Ordered Local Coloring	C&P Ordered Start and Ordered Local Coloring	C&P with Ordered Saturation Start & Ordered Local Coloring	C&P with Ordered DSatur Start & Ordered Local Coloring
brock400_2	1246	1134	1044	1092
brock400_4	1467	282	287	252
brock800_2	22,121	24,626	24,466	26,148
brock800_4	10,938	15,228	13,038	12,761
keller5	270,687	278,555	144,558	159,936
p_hat300-3	151	188	132	66
p_hat700-2	1721	1979	659	355
p_hat1500-1	11	11	12	12

2.2. Evaluation of Exact Algorithms with a Coloring Bound for Use in HS2

Although the results in Tables 1 and 2 give a useful comparison, it should be noticed that the subproblems to which exact and pseudoexact search is typically applied in HTS may be much smaller than the instances used in those tables. Thus, four of the algorithms were evaluated further by using them in place of the exact and pseudoexact search in HTS. The critical improvement in performance arises from maximizing the mean number of tabu search iterations per second, and this was calculated for an extended run on each of seven instances. These were DIMACS, BHOSLIB, and permutation code instances used in [2], and, in general, the same parameters were used for HTS as in the experiments in [2]. The results are shown in Table 3.

It appears that there is much less difference in the results than in the previous tables, in part because if the time for exact search is small the time to calculate neighborhoods becomes more significant. However, it appears from further experiments that the third and fourth algorithms (using ordering) allow the parameter *Tssetmin* in HTS (determining the smallest current clique from which vertices may be removed) to be smaller. These smaller values allow exact search to be applied to larger subproblems without excessively slowing tabu search and can improve performance as a result. From inspection of Table 3, the third algorithm "C&P with ordered local coloring" was selected for use in an improved version HTS2 of HTS. This algorithm also had the merit that it was simpler than the fourth algorithm. Pseudocode for "C&P with ordered local coloring" is shown in Algorithm 1.

Table 3. Evaluation of the mean number of tabu search iterations per second for four of the algorithms appearing in Tables 1 and 2. These replace exact search and pseudoexact search in Hybrid Tabu Search (HTS). The best exact algorithm should maximize the number of tabu search iterations per second during an extended run.

Instance	C&P	C&P Start & Local Coloring	C&P with Ordered Local Coloring	C&P with Ordered DSatur Start and Ordered Local Coloring
C4000_5	1659	1725	1795	1389
C2000_9	23	252	236	215
keller6	260	225	285	329
frb35-17-4	833	2359	2827	1660
frb50-23-3	765	825	996	784
frb50-23-2	763	796	948	731
7_5	79	98	96	94

Algorithm 1: The exact algorithm "C&P with ordered local coloring" applied to a graph or subgraph with vertex set V and edge set E.

Require A set of selected vertices $F \subseteq V$ and a set of potential expansion vertices $S \subseteq V$. The best clique retrieved so far is contained in *Best* and *External_lower_bound* is a supplied external value. The recursive algorithm is invoked with $F := \emptyset$, $S := V$ and the global variable $Best := \emptyset$.

Exact(F**,** S**)**

while $S \neq \emptyset$ **do**
  if $|F| + |S| > \max\{|Best|, External\_lower\_bound - 1\}$ **then**
    color S greedily, using smallest available color;
    sort S in non-increasing order of color;
    select $s \in S$ in above order;
    $S := S \backslash \{s\}$;
    $S' := S$;
    for $z \in S'$ **do**
      if $(z, s) \notin E$ **then** { if (z, s) is not an edge }
        $S' := S' \backslash \{z\}$;
      end if
    end for
    $F' := F \cup \{s\}$;
    if $S' = \emptyset$ and $|F'| > |Best|$ **then**
      $Best := F'$;
    else
      determine number of colors $numcolors(S')$ used in S';
      if $|F'| + numcolors(S') > \max\{|Best|, External\_lower\_bound - 1\}$ **then**
        Exact(F', S', $Best$);
      end if
    end if
  end if
end while
return(Best)

3. Minor Improvements in HTS2

Apart from the change from the exact and pseudoexact search in HTS to the exact "ordered local coloring" algorithm detailed in Algorithm 1, there are some other relatively minor changes from HTS in HTS2. As well as the tabu search algorithm, HTS has a main heuristic that generates good starting solutions (sometimes finds the best solution itself) and i-optimizations for $i = 1, 2, 3$. These i-optimizations remove i vertices from the current clique in all possible ways and apply the

exact algorithm to the neighbor set of vertices adjacent to all remaining vertices in an attempt to find a new larger clique. This continues until there is no improvement. In HTS, there are thresholds θ_1, θ_2, θ_3, θ_{ts} for the four optimizations. In HTS, these thresholds are allowed to be different and are selected so that each optimization is only applied to very promising cases, and 3-optimization never makes HTS unacceptably slow. In HTS2, these four thresholds are replaced by a single threshold θ. The justification for this is that the improved performance of the tabu search makes a different threshold for tabu search unnecessary, and the relative slowness of 3-optimization in some instances can be dealt with by considering the size of the current clique. Thus, the overall structure of HTS2 is as shown in Algorithm 2.

Algorithm 2: Outline of the overall structure of the algorithm HTS2.

```
BestC := ∅;
k := 0;
while runtime ≤ max_runtime do
    k := k + 1;
    C := Main_heuristic(k);
    if |C| ≥ θ then
        1_optimize(C);
        C := ts_optimize(C);
        if |C| ≤ 200 then
            C := 3_optimize(C);
        else
            C := 2_optimize(C);
        end if
    end if
    if |C| > |BestC| then
        BestC := C;
    end if
end while
```

For any current clique C, $N(C)$ denotes the set of vertices of V adjacent to all vertices in C. In the main heuristic of the original HTS algorithm, a sequence S is selected. A variable *adjchoices* cycles through the values in S and controls the choice of the next vertex in $N(C)$ to add to the clique C. A set U consists of all vertices of $N(C)$ if $|N(C)| \leq adjchoices$, or *adjchoices* randomly selected vertices of $N(C)$ otherwise. A vertex u of U is chosen to add to C which gives the largest value of $N(C \cup \{u\})$. The most common S in the experiments in [2] is $S = [1, 50, 80, 100, 120, 160, 200, 300, 600, 800]$, and S is fixed to this choice in HTS2 unless the graph has < 1000 vertices when $S = [1, 50, 80, 100, 120, 160, 200, 300]$. Thus, no decision on an appropriate S is necessary in HTS2. Pseudocode for the main heuristic is presented in Algorithm 3, and pseudocode for the i-optimizations is presented in Algorithm 4.

In the original HTS algorithm a parameter, *Nontabu max* is used in tabu search. If the neighborhood to which exact search is applied has at least *Nontabu max* vertices, a protection mechanism is invoked (selecting a random vertex in the neighborhood) to avoid application of the exact search to an excessively large neighborhood. Typical values of 30 or 60 were used. In HTS2, a parameter λ_2 from the main heuristic, to be outlined in the next section, is used instead of *Nontabu max*.

A final change is to replace the condition that the tabu search optimization runs for *Tstime* seconds unless the size of the clique increases, by the requirement that it runs for 10,000 iterations, unless the size of the clique increases. This avoids the need for the parameter *Tstime* and proved satisfactory for all instances. Pseudocode for the revised tabu search algorithm is presented in Algorithm 5.

Algorithm 3: Main heuristic at iteration k.

The algorithm is invoked for a graph $G = (V, E)$ with $C :=${random vertex in V}. Parameters supplied are the threshold θ from Algorithm 2, and two further thresholds λ_1, λ_2 for exact search. The sequence of values $S = [s_i]$ is also supplied. Following the call to the main heuristic, C contains the best clique found by the main heuristic at iteration k.

Main_heuristic(k)

$C :=${random vertex in V};
$Exactstored := False$; $adjchoices := s_{1+k \bmod |S|}$;
while $|N(C)| > 0$ **and** $|N(C)| \geq \theta - |C|$ **do**
  if $|N(C)| < \lambda_2$ **then**
    $Best := \emptyset$; $External\_lower\_bound := \theta - |C|$;
    $Exact(\emptyset, N(C))$;
    $C := C \cup Best$;
  else
    if $|N(C)| < \lambda_1$ **and** $Exactstored = False$ **then**
      $C_{\text{stored}} := C$; $N(C)_{\text{stored}} := N(C)$;
      $Exactstored := True$;
    end if
    if $|N(C)| \leq adjchoices$ **then**
      $U := N(C)$;
    else
      $U := \{v_{j_1}, v_{j_2}, \ldots, v_{j_{adjchoices}} | v_{j_i} \in N(C)$ selected randomly$\}$;
    end if
    Choose $u \in U$ such that $|N(C \cup \{u\})|$ is maximal;
    $C := C \cup \{u\}$;
  end if
end while
if $Exactstored = True$ **and** $|C| \geq \theta$ **then**
  $Best := \emptyset$; $External\_lower\_bound := \theta - |C_{\text{stored}}|$;
  $Exact(\emptyset, N(C)_{\text{stored}})$;
  $C_{\text{temp}} := C_{\text{stored}} \cup Best$;
  if $|C_{\text{temp}}| > |C|$ **then**
    $C := C_{\text{temp}}$;
  end if
end if
return(C);

Algorithm 4: i_Optimize.

i_Optimize(C)

$Current\_best := C$;
Recursive_i_opt(C)
for all $S \subset C$ with $|S| = i$ **do**
  if $|N(C \setminus S)| > i$ **then**
    $Best := \emptyset$; $External\_lower\_bound := i + 1$;
    Exact($\emptyset$, $N(C \setminus S)$);
    if $|Best| > i$ **then**
      $Current\_opt := Best \cup (C \setminus S)$
      if $|Current\_opt| > |Current\_best|$ **then**
        $Current\_best := Current\_opt$;
        Recursive_i_opt($Current\_opt$);
      end if
    end if
  end if
end for
return($Current\_best$);

Algorithm 5: Tabu search.

ts_Optimize(*C***)**;
$Ts\_count := 0$;
$Tabulist := \emptyset$; $Tscurrent := C$; $Tsbest := C$;
while ($Ts\_count < 10000$) **do**
  $Ts\_count := Ts\_count + 1$; $Aspiration := False$;
  for all $v \in Tscurrent$ **do**
    $Best := \emptyset$; $External\_lower\_bound := |Tsbest| - |Tscurrent| + 1$;
    Exact($\emptyset$, $N(Tscurrent \setminus \{v\})$);
    $Tstemp1 := Best \cup (Tscurrent \setminus \{v\})$;
    Update a list L_1 of all $v \in Tscurrent$ with $|Tstemp1|$ maximal (and for this value of $|Tstemp1|$ the value $|N(Tscurrent \setminus \{v\})|$ is maximal);
    if $|Tstemp1| > |Tsbest|$ **then**
      $Aspiration := True$;
    else
      $Nontabu := \{w \in N(Tscurrent \setminus \{v\}) | w \notin Tabulist\}$;
      $Best := \emptyset$;$External\_lower\_bound := 0$;
      Exact($\emptyset$, $Nontabu$);
      $Tstemp2 := Best \cup (Tscurrent \setminus \{v\})$;
      Update a list L_2 of all $v \in Tscurrent$ with $|Tstemp2|$ maximal (and for this value of $|Tstemp2|$ the value $|Nontabu|$ is maximal);
    end if
  end for
  if Aspiration = true **then**
    select $v' \in L_1$ randomly;
    $Best := \emptyset$; $External\_lower\_bound := |Tsbest| - |Tscurrent| + 1$;
    Exact($\emptyset$, $N(Tscurrent \setminus \{v'\})$);
    $Tscurrent := Best \cup (Tscurrent \setminus \{v'\})$; $Tsbest := Tscurrent$;
    $Ts\_count := 0$;
  else
    Select $v'' \in L_2$ randomly;
    $Tscurrent := Tscurrent \setminus \{v''\}$;
    Add v'' to $Tabulist$ (removing oldest entry if list length exceeds $Tstenure$);
    if $|Tscurrent| \leq Tssetmin + 1$ **then**
      $Nontabu := \{w \in N(Tscurrent) | w \notin Tabulist\}$;
      if $|Nontabu| < \lambda_2$ **then**
        $Best := \emptyset$; $External\_lower\_bound := 0$;
        Exact($\emptyset$, $Nontabu$);
        $Tscurrent := Best \cup Tscurrent$;
      else
        Select $v''' \in Nontabu$ randomly;
        $Tscurrent := \{v'''\} \cup Tscurrent$;
      end if
    end if
  end if
end while
if $|Tsbest| > |C|$ **then**
  $C := Tsbest$;
end if
return(C);

4. Simplification of Parameters

There are several parameters used in HTS. These are not difficult to determine by a short exploratory run, but it is more convenient if most of them are determined by the algorithm itself. The parameters in question are θ_1, θ_2, θ_3, θ_{ts}, λ_1, λ_2, $Tstime$, $Tstenure$, and $Tssetmin$. Pseudoexact search in HTS also has a number of coefficients, but these are no longer relevant as pseudoexact search is not used in HTS2. The use of $Tstime$ was replaced by a condition "10,000 iterations without improvement" in the previous section.

The four θ parameters in HTS are replaced in HTS2 by a single parameter θ (see Algorithm 2) determined as follows. The main heuristic is run 100 times. If the largest clique C_{max} found is not unique, then $\theta = |C_{max}|$. If this value is unique, then θ is the number of vertices in the second largest clique. This ensures that the other optimizations are only applied to the most promising cliques generated, but avoids single outliers that might only be generated very rarely in later iterations.

The two parameters λ_1 and λ_2 are used in the main heuristic (see Algorithm 3). If the size of the neighborhood $N(C)$ of the current clique satisfies $|N(C)| < \lambda_2$, then the generation of the clique by the main heuristic is completed by applying the exact algorithm to $N(C)$. If the clique generated has at least θ vertices, the algorithm reverts to a somewhat larger stored neighborhood with less than λ_1 vertices and applies the exact algorithm. In HTS2, the two λ parameters are determined by the edge density $2|E|/(|V|(|V|-1))$, as shown in Table 4. The parameter λ_2 is also used in the protection mechanism within tabu search to avoid applying the exact algorithm to excessively large subproblems.

Table 4. Choices of parameters λ_1 and λ_2.

1	$\geq$ edge density	≥ 0.95	$\lambda_1 = 80$	$\lambda_2 = 60$
0.95	$>$ edge density	≥ 0.9	$\lambda_1 = 100$	$\lambda_2 = 80$
0.9	$>$ edge density	≥ 0.84	$\lambda_1 = 140$	$\lambda_2 = 120$
0.84	$>$ edge density	≥ 0.74	$\lambda_1 = 160$	$\lambda_2 = 140$
0.74	$>$ edge density		$\lambda_1 = 180$	$\lambda_2 = 160$

The value of the tabu tenure *Tstenure* is not at all critical for HTS2. Values of 6, 12, and 20 were used, and all proved successful.

The tabu search parameter *Tssetmin* is the critical parameter to be set by the user. It must be less than θ, but, if it is too small, attempts may be made to solve exactly subproblems that are too large, requiring the protection mechanism. Otherwise, larger values make tabu search faster, but smaller values may make tabu search more powerful. A balance should be achieved.

It should be noted that the current implementation allows the original θ and λ parameters from HTS to override these choices if the user requires it. Otherwise, *Tstenure* and *Tssetmin* are the only relevant parameters in HTS2.

5. Comparison of HTS, HTS2, and SBTS

This section presents a comparison of the original HTS algorithm, the new HTS2 algorithm, and SBTS. The software for the SBTS algorithm is available online as [17]. The benchmarks selected are BHOSLIB instances [18], some of the harder DIMACS instances [16], Sloane code construction instances [6], and permutation code construction instances from [7]. A few instances shown in [2] to be most easily solved using a pre-processing method are excluded, so the description of pre-processing need not be repeated here. These benchmarks are not ideal in the sense that the optimal solution is known for many of them, but they are at least easily available to allow comparison by others. Using an exact algorithm cannot be expected to be the fastest algorithm, but the more thorough local search might lead to improved solutions if improved solutions are possible. Of particular interest, then, is the permutation code instance 7_5. The vertices correspond to the permutations on 7 symbols, excluding those permutations at Hamming distance 1, 2, 3, or 4 from the identity permutation. Two vertices are adjacent if they are at Hamming distance ≥ 5. A theoretical construction using group theory shows that there exists a clique with 78 vertices [19], but the largest clique found with a maximum clique algorithm previously has 72 vertices (or 73 using the pre-processing method in [2]). For permutation code instances other than 7_4 and 7_5, the vertices of the graph correspond to orbits of permutations under a group, as explained in [7].

The processor used for the experiments was an Intel(R) Core(TM) i3-9100 3.60GHz processor with 4 GB of RAM. In most cases, the parameters used for HTS were the same as given in [2], the parameters for *Tstenure* and *Tssetmin* in HTS2 were the same as used for these parameters in HTS in [2], and the default setting for tabu tenure was used in SBTS. Results for BHOSLIB instances are given in Table 5,

and results for all other instances are given in Table 6. In these two tables, the first column gives the instance, and the second column gives the number of vertices in the largest known clique (marked with an asterisk if the clique is known to be optimal). The third, fourth, and fifth columns give the time in seconds to find the solution in column 2, with a note of the number of vertices found if the result in column 2 is not achieved.

Table 5. Comparison of run times to find optimal solution of BHOSLIB instances for HTS, HTS2, and SBTS (Swap-Based Tabu Search). * denotes that the best known result is optimal

Instance	Optimal or Best Known Number of Vertices	HTS (secs)	HTS2 (secs)	SBTS (secs)
frb30-15-1	30 *	20	13	< 1
frb30-15-2	30 *	3	< 1	< 1
frb30-15-3	30 *	16	38	< 1
frb30-15-4	30 *	14	1	< 1
frb30-15-5	30 *	38	24	< 1
frb35-17-1	35 *	175	53	< 1
frb35-17-2	35 *	7	11	< 1
frb35-17-3	35 *	29	2	< 1
frb35-17-4	35 *	807	129	2.7
frb35-17-5	35 *	138	9	1.2
frb40-19-1	40 *	20	1	< 1
frb40-19-2	40 *	555	123	25.7
frb40-19-3	40 *	77	1	7.2
frb40-19-4	40 *	1394	32	1.1
frb40-19-5	40 *	6804	2000	28.1
frb45-21-1	45 *	9692	5778	16.0
frb45-21-2	45 *	57,227	379	5.5
frb45-21-3	45 *	93,925	2568	16.7
frb45-21-4	45 *	12,123	1690	31.6
frb45-21-5	45 *	111,435	2173	18.0
frb50-23-1	50 *	22,883	10,649	67.0
frb50-23-2	50 *	90,753	1219	571.4
frb50-23-3	50 *	196,389	17,472	2098.4
frb50-23-4	50 *	119	295	54.4
frb50-23-5	50 *	563	350	80.8

Table 6. Comparison of run times to find optimal or best known solution of Sloane, DIMACS, and permutation code instances for HTS, HTS2, and SBTS. If the solution in the second column is not achieved, the best solution found is indicated in brackets. * denotes that the best known result is optimal

Instance	Optimal or Best Known Number of Vertices	HTS (secs)	HTS2 (secs)	SBTS (secs)
1dc.1024	94 *	11,899	8	< 1
1dc.2048	172	20,561	324	< 1
1et.1024	171 *	25,988	1485	< 1
1et.2048	316 *	1366	5219	119.5
1tc.1024	196 *	1309	69	< 1
1tc.2048	352 *	797	19,786	773.5
1zc.1024	112	8880	445	2.7
1zc.2048	198	15,477	2366	23.6
2dc.1024	16 *	51	< 1	< 1
2dc.2048	24 *	131	34	< 1

Table 6. *Cont.*

Instance	Optimal or Best Known Number of Vertices	HTS (secs)	HTS2 (secs)	SBTS (secs)
keller6	59	1057	1259	158.4
C4000_5	18 *	390	312	< 1
C2000_9	80	658 (78 ⋄)	140,054	334,910 ♭
7_4	349	1,353,356	258,579	21,336
7_5	78	102,394 (73 †)	364,585 (75 ♯)	215,270 (72 ‡)
8_5 cyclic	49	5163	6799	24,326.0 (46 ♮)
8_5 ext. cyclic	88	182	1	25.6
10_4	209	1257	3228	59.0
10_5	26	4	13	189.9
11_4	220	737	46	4.4
11_5	26	3	7	< 1
12_4	220	430	2	112.0
12_5	25	1	< 1	2

⋄ A clique with 80 vertices was found by HTS in [2] using a pre-processing method. ♭ Jin and Hao [3] report a much smaller time for successful runs, but only 2% of 100 runs were successful. † 73 was found in 59,788 s but not improved in 906,471 s. ♯ In addition, smaller cliques with 74 vertices were found 4 times in 364,585 s. ‡ 72 was found in 215,270 s but not improved in 1,399,916.1 s. ♮ 46 was found in 24,326.0 s but not improved in 61,097.2 s.

For most instances, HTS2 is faster than HTS and finds a larger clique for C2000_9 than HTS without pre-processing, and also a larger clique for 7_5 than HTS with or without pre-processing. For BHOSLIB, most DIMACS and Sloane code construction instances, SBTS is much faster than HTS2, as might be expected. However, HTS2 is more competitive on harder DIMACS instances, such as C2000_9, and performs much better than SBTS on 7_5 and 8_5 cyclic. For 7_5, HTS2 found a clique with 75 vertices once and cliques with 74 vertices four times in 364,585 s, whereas SBTS found cliques with 72 vertices twice but could not improve this in 1,399,916.1 s. Similarly, for 8_5 cyclic, HTS2 found a clique with 49 vertices in 6799 s, whereas SBTS found cliques with 46 vertices four times but could not improve this in 61,097.2 s.

6. Conclusions

The replacement of the pseudoexact algorithm in HTS with an improved exact algorithm in HTS2 has allowed a pure comparison of tabu search using exact search with a more standard type of tabu search. The interest of the authors is in code construction problems, where certain instances only have to be solved once. The critical issue is that the largest clique possible is found, and run time is relatively unimportant. It is clear that SBTS is much faster than HTS2 for most instances and should be used in applications where clique problems have to be solved quickly. HTS2 has found the same size clique in all instances except 7_5 and 8_5 cyclic, where it has found larger cliques with three more vertices. The instance 7_5 is an important addition to the set of standard benchmarks for maximum clique problems, as it is of a similar size to the largest benchmarks, but no algorithm has been able to find a clique with 78 vertices that is known to exist. In summary, HTS2 finds the same size clique as SBTS for 46 instances, more slowly for 40 instances but faster for 4 instances. It finds a larger clique than SBTS for two instances, while SBTS never finds a larger clique than HTS2.

Supplementary Materials: The following are available online at www.mdpi.com/xxx/s1, Code for HTS2 is available in the Supplementary Material.

Author Contributions: Development and presentation of original HTS algorithm, D.H.S., S.P. and R.M.; conceptualization of graph coloring approach, D.H.S. and R.M.; software development, D.H.S.; permutation code benchmarks, D.H.S. and R.M., results, D.H.S.; presentation, D.H.S., S.P. and R.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Montemanni, R., Barta J. and Smith, D.H. Graph Colouring and Branch and Bound Approaches for Permutation Code Algorithms. In *New Advances in Information Systems and Technologies. Advances in Intelligent Systems and Computing*, Rocha, Á., Correia, A., Adeli, H., Reis, L., Mendonça Teixeira, M., Eds.; *444*. Springer: Cham, Switzerland, 2016.
2. Smith, D.H.; Perkins, S.; Montemanni, R. Solving the maximum clique problem with a hybrid algorithm. *Int. J. Metaheuristics* **2019** *7*, 152–175.
3. Jin, Y.; Hao, J.-K. General swap-based neighborhood tabu search for the maximum independent set problem. *Eng. Appl. Artif. Intel.* **2015**, *37*, 20–33.
4. Wu, Q.; Hao, J.-K.. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.* **2015**, *242*, 693–709.
5. Leese, R.; Hurley, S. (Eds.) *Methods and Algorithms for Radio Channel Assignment*; Oxford University Press: Oxford, UK, 2002.
6. Sloane, N.J.A. Challenge Problems: Independent Sets in Graphs. Available online: http://oeis.org/A265032/a265032.html (accessed on 17 June 2020).
7. Smith, D.H.; Montemanni, R. A new table of permutation codes. *Design. Code. Cryptogr.* **2012**, *63*, 241–253.
8. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*; Miller, R.E., Thatcher, J.W., Eds.; Plenum Press: New York, NY, USA, 1972; pp. 85–103.
9. Fallah, M.K.; Keshvari, V.S.; Fazlali, M. A parallel hybrid genetic algorithm for solving the maximum clique problem. In: *High-Performance Computing and Big Data Analysis. TopHPC 2019*; Grandinetti, L., Mirtaheri, S., Shahbazian, R., Eds.; Springer, Cham. Switzerland, 2019; Volume 891, doi:10.1007/978-3-030-33495-6_29
10. Guo, P.; Wang, X.; Zeng, Y.; Chen, H. MEAMCP: A membrane evolutionary algorithm for solving maximum clique problem. *IEEE Access* **2019**. Availabel online: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8788505 (accessed on 24 September 2020).
11. Tayachi, D.; Khemiri, M. Solving the maximum clique problem using a hybrid particle swarm optimization algorithm. *Int. J. Operat. Res. Inf. Sys.* **2018**, *9*, 21– 35.
12. Kiziloz, ; H.E.; Dokeroglu, T. A robust and cooperative parallel tabu search algorithm for the maximum vertex weight clique problem. *Comput. Indust. Eng.* **2018**, *118*, 54–66.
13. Djeddi, Y.; Haddadene, H.A.; Belacel, N. An extension of adaptive multi-start tabu search for the maximum quasi-clique problem. *Comput. Indust. Eng.* **2019**, *132*, 280–292.
14. Carraghan, R.; Pardalos, P.M. An exact algorithm for the maximum clique problem. *Oper. Res. Lett.* **1990**, *9*, 375–382.
15. Brélaz, D. New methods to color the vertices of a graph. *Commun. ACM* **1979**, *22*, 251–256.
16. DIMACS: Clique Benchmark Instances. Available online: http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark (accessed on 17 June 2020).
17. Jin, Y.; Hao, J.-K. SBTS Software. Available online: http://www.info.univ-angers.fr/pub/hao/mis.html (accessed on 17 June 2020).
18. BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems. Available online: http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark (accessed on 17 June 2020).
19. Janiszczak, I.; Lempken, W.; Östergård, P.R.J.; Staszewski, R. Permutation codes invariant under isometries. *Design. Code. Cryptogr.* **2015**, *75*, 497–507.