Computers & Operations Research ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Reactive and dynamic local search for max-clique: Engineering effective building blocks

Roberto Battiti*, Franco Mascia

Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento, Via Sommarive, 14 – I-38100 Povo (Trento), Italy

ARTICLE INFO

Keywords:
Hybrid metaheuristics
Reactive local search
Dynamic local search
Stochastic local search
Algorithm engineering

ABSTRACT

This paper presents the results of an ongoing investigation about how different algorithmic building blocks contribute to solving the maximum clique problem. We consider greedy constructions, plateau searches, and more complex schemes based on dynamic penalties and/or prohibitions, in particular the recently proposed technique of dynamic local search and the previously proposed reactive local search (RLS). We design a variation of the original RLS algorithm where the role of long-term memory (LTM) is increased (RLS-LTM). In addition, we consider in detail the effect of the low-level implementation choices on the CPU time per iteration. We present experimental results on randomly generated graphs with different statistical properties, showing the crucial effects of the implementation, the robustness of different techniques, and their empirical scalability.

1. Prohibition- and penalty-based methods for maximum clique

The availability of heuristic solution techniques for relevant combinatorial problems is now large and the scientific and practical issues arise of tuning, adapting, combining and hybridizing the different techniques. In hybrid metaheuristics, the potential for reaching either better average results, or more robust results with less variability is large, but the task is complex. First of all, if one naively tries to consider all possible combinations of component and parameters a combinatorial explosion occurs. Secondly, like in all scientific challenges, one aims not only at beating the competition on specific benchmarks, but also at understanding the contribution of the different parts to the whole and at discriminating the basic principles for achieving successful hybrids.

The present investigation is focussed on algorithmic components used to solve the maximum clique (MC) problem in graphs with state-of-the-art results. In particular it is focussed on combining methods based on local search with memory-based complements to achieve a proper balance of intensification and diversification during the search. The considered paradigms are of (i) using prohibitions to achieve diversification and avoid small cycles in the search trajectory (limit cycles or the equivalent of "chaotic attractors" in discrete dynamical systems), (ii) using restarts triggered by events happening during the search, and (iii) using modifications of the objective function to influence the trajectory and achieve diversification by modifying the fitness surface instead of reducing the number of admissible (non-prohibited) moves.

In addition, stochastic local search (SLS) engineering methods are considered to develop efficient implementations of the single steps of the SLS-based techniques, by considering data structures to support the choice of the next move and the use of memory during the search. Let us briefly summarize the considered techniques and the concentration of this work.

The MC problem in graphs is a paradigmatic combinatorial optimization problem with relevant applications [13], including information retrieval, computer vision, and social network analysis. Recent interest includes computational biochemistry, bioinformatics and genomics, see, for example [8,11]. The problem is NP-hard and strong negative results have been shown about its approximability [9], making it an ideal testbed for search heuristics. Let $G=(V,E)$ be an undirected graph, $V=\{1,2,\ldots,n\}$ its vertex set, $E\subseteq V\times V$ its edge set, and $G(S)=(S,E\cap S\times S)$ *the subgraph induced by* S, where S is a subset of V. A graph $G=(V,E)$ is *complete* if all its vertices are pairwise adjacent, i.e., $\forall i,j\in V$, $(i,j)\in E$. A *clique* K is a subset of V such that $G(K)$ is complete. The MC problem asks for a clique of maximum cardinality.

Reactive search [4,5,15] advocates the use of *machine learning* to automate the parameter tuning process and make it an integral and fully documented part of the algorithm. Learning is performed online, and therefore *task-dependent and local properties* of the configuration space can be used. A reactive local search (RLS) algorithm for the solution of the MC problem is proposed in [3,6]. RLS is based on local search complemented by a feedback (history-sensitive) scheme to determine the amount of diversification. The reaction acts on the

* Corresponding author.
E-mail address: battiti@disi.unitn.it (R. Battiti).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.02.013

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

single parameter that decides the temporary *prohibition* of selected moves in the neighborhood. The performance obtained in computational tests appears to be significantly better with respect to all algorithms tested at the second DIMACS implementation challenge (1992/1993).[1]

Recently, a SLS algorithm (DLS-MC) is developed in [14]. It is based on a clique expansion phase followed by a plateau search after a MC is encountered. Diversification uses vertex *penalties* which are dynamically adjusted during the search, a "forgetting" mechanism decreasing the penalties is added, and vertex degrees are not considered in the selection. The authors report a very good performance on the DIMACS instances after a preliminary extensive optimization phase to determine the optimal penalty delay (*pd*) parameter for each instance. While the number of iterations (additions or deletions of nodes to the current clique) is in some cases larger than that of competing techniques, the small complexity of each iteration when the algorithm is realized through efficient supporting data structures leads to smaller overall CPU times.

The motivation of this work is threefold. First, we want to investigate how the different algorithmic building blocks contribute to effectively solving MC instances corresponding to random graphs with different statistical properties. In particular, the investigation considers the effects of using the vertex degree information during the search, starting from simple to more complex techniques. Second, we want to assess how different implementations of the supporting data structures affect CPU times. For example, it may be the case that larger CPU times are caused by using a high-level language implementation w.r.t. low-level "pointer arithmetic". Having available the original software simplified the starting point for this analysis. Third, the DIMACS benchmark set (developed in 1992) has been around for more than a decade and there is a growing risk that the desire to get better and better results on the same benchmark will bias the search of algorithms in an unnatural way. We therefore decided to concentrate the experimental part on two classes of random graphs, chosen to assess the effect of degree variability on the effectiveness of different techniques. Last but not least, a new algorithmic version has been developed, RLS or RLS-LTM, which maintains the complete trajectory in its data structure.

2. Algorithmic building blocks of increasing complexity

In local search algorithms for MC, the basic moves consist of the addition to or removal of single nodes from the current clique. A swap of nodes can be trivially decomposed into two separate moves. The local changes generate a search trajectory $X^{\{t\}}$, the current clique at different iterations t. Two sets are involved in the execution of basic moves: the set of the *improving neighbors* POSSIBLEADD which contains nodes connected to all the elements of the clique, and the set of the *level neighbors* ONEMISSING containing the nodes connected to all *but* one element of the clique, see Fig. 1. The various simple building blocks considered are named following the *BasicScheme—CandidateSelection* structure. The *BasicScheme* describes how the greedy expansion and plateau search strategies are combined, possibly with prohibitions or penalties. The *CandidateSelection* specifies whether the vertex degree information is used during the selection of the next candidate move. If it is used, there are two possibilities: of using the static node degree in G or the dynamic degree in the subgraph induced by the POSSIBLEADD set.

2.1. Repeated expansions

The starting point for many schemes is given by the expansion of a clique after starting from an initial seed vertex. At each iteration

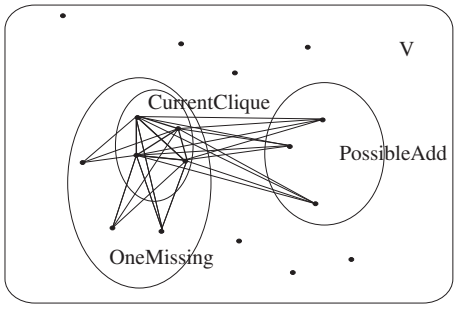

Fig. 1. Neighborhood of the current clique.

```
1. EXP-RAND (maxIterations)
2.   iterations ← 0
3.   while iterations < maxIterations do
4.     C ← random v ∈ V
5.     EXPAND (C)

6. EXPAND (C)
7.   while POSSIBLEADD ≠ ∅ do
8.     C ← C ∪ random v ∈ POSSIBLEADD
9.     iterations ← iterations + 1
```

Fig. 2. Greedy expansion algorithm.

the next vertex to be added can be chosen from the POSSIBLEADD set through different levels of "greediness" when one considers the vertex degrees:

- *Exp-Rand.* The node is selected at random among the possible additions. When a MC is encountered one restarts from a random node. The pseudo-code is shown in Fig. 2.
- *Exp-StatDegree.* At each iteration, a random node is chosen among the candidates having the highest degree in G. The EXPAND subroutine in Fig. 2 is modified by substituting line 8 with:
- $C \leftarrow C \cup$ {*random* $v \in$ POSSIBLEADD such that $deg_G(v)$ *is maximum*}.
- *Exp-DynDegree.* In this version, the selection of the candidate is not based on the degree of the nodes in G, but on the degree in POSSIBLEADD. This greedy choice will maximize the number of nodes remaining in POSSIBLEADD after the last addition. Line 8 becomes:
- $C \leftarrow C \cup$ {*random* $v \in$ POSSIBLEADD s.t. $deg_{\text{POSSIBLEADD}}(v)$ *is max*}.

2.2. Expansion and plateau search

This algorithm alternates between a greedy expansion and a plateau phase, choosing between the possible candidate nodes with different ways to consider the vertex degrees:

- *ExpPlat-Rand.* During the *expansion* phase, new vertices are chosen randomly from POSSIBLEADD and moved to the current clique. When POSSIBLEADD is empty and therefore no further expansion is possible, the *plateau* phase starts. In this phase, a node belonging to the *level neighborhood* ONEMISSING is swapped with the only node not connected to it in the current clique. The *plateau* phase does not increment the size of the current clique and it terminates as soon as there is at least an element in the POSSIBLEADD set, or if no candidates are available in ONEMISSING. As it is done in [14], nodes cannot be selected twice in the same plateau phase. In order

[1] http://dimacs.rutgers.edu/Challenges/

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

```
1.  EXPPLAT-RAND (maxIterations, maxPlateauSteps)
2.  ⌈ iterations ← 0
3.  | while iterations < maxIterations do
4.  | ⌈ C ← random v ∈ V
5.  | | while POSSIBLEADD ≠ ∅ do
6.  | | ⌈ EXPAND (C)
7.  | | ⌊ PLATEAU (C, maxPlateauSteps)

8.  PLATEAU (C, maxPlateauSteps)
9.  ⌈ count ← 0
10. | while POSSIBLEADD = ∅ and ONEMISSING ≠ ∅
11. | and count < maxPlateauSteps do
12. | ⌈ C ← C ∪ random v ∈ ONEMISSING
13. | | remove from C the node not connected to v
14. | | iterations ← iterations + 2
15. | ⌊ count ← count + 1
```

Fig. 3. EXPPLAT-RAND. algorithm, alternating between expand and plateau phases.

to avoid infinite loops, the number of plateau searches is limited to *maxPlateauSteps*. Starting from EXP-RAND, the base algorithm is adapted to deal with the alternation of the two phases, see Fig. 3. Let us note that, if PLATEAU returns with POSSIBLEADD$\neq\emptyset$ then a new expansion is tried as described in lines 5–7. The iterations are incremented by 2 during a swap because it is counted as a deletion followed by an addition.

- *ExpPlat-StatDegree.* This algorithm is a modified version of EXPPLAT-RAND (Fig. 3) with the static degree selection during the expansion and the plateau.
- *ExpPlat-DynDegree.* This algorithm is the same of EXPPLAT-STATDEGREE, apart from the selection based on the dynamic degree during the expansion phase.

2.3. Algorithms based on penalties or prohibitions

More complex schemes can be obtained by using diversification strategies to encourage the search trajectories to visit unexplored regions of the search space. These methods are particularly effective for "deceptive" instances [7], where the sub-optimal solutions attract the search trajectories.

- *ExpPlatProhibition-Rand.* A simple diversification strategy can be obtained by prohibiting selected moves in the neighborhood. In detail, after a node is added or deleted from the current clique, the algorithm prohibits moving it for the next T iterations. Prohibited nodes cannot be considered among the candidates of expansion and plateau phases. When all the moves are prohibited a restart is performed.
- *DLS-MC.* To achieve diversification during the search, penalties are assigned to vertices of the graph [14]. The algorithm alternates between expansion and plateau phases. Selection is done by choosing the best candidate among the set of the nodes in the neighborhood having minimum penalty.
- When the algorithm starts, the penalty value of every node is initialized to 0 and when no further expansion or plateau moves are possible, the penalties of nodes belonging to the clique are incremented by one. All penalties are decremented by one after *pd* restarts, see [14] for additional details and results.
- *RLS.* This algorithm alternates between expansion and plateau phases, like DLS-MC, but it selects the nodes among the non-prohibited ones which have the highest degree in POSSIBLEADD. The prohibition time is adjusted reactively depending on the search history. In the "history" a fingerprint of each configuration is saved in a hash table. Restarts are executed only when the algorithm cannot improve the current configuration within a fixed number of iterations, see [6] for details.
- To allow for a comparison between the different amount of "greediness" in the node selection, a modification of RLS is introduced (called RLS-StatDegree) which uses the static degree instead of the dynamic degree selection.

The research presented in this paper considers two kinds of changes to the original RLS version. The first changes are algorithmic and influence the search trajectory, while the second one refers only to the more efficient implementation of the supporting data structures, with no effect on the dynamics. The algorithmic changes are the following ones. In the previous version, the search history was cleared at each restart, now, in order to allow for a more efficient diversification, the entire search history is kept in memory. To underline this fact, the new version is called RLS or RLS-LTM. Having a longer memory caused the parameter T to explode on some specific instances characterized by many repeated configuration during the search. If the prohibition becomes much larger than the current clique size, after a MC is encountered and one node has to be extracted from the clique, all other nodes will be forced to leave the clique before the first node is allowed to enter again. This may cause spurious oscillations in the clique membership which may prevent discovering the globally optimal clique. An effective way to avoid the above problem is to put an upper-bound *MAX_T* equal to a proportion of the current estimate of the MC. More specifically, the upper-bound is set to $|\text{CLIQUE}| * 0.5$.

3. Computational experiments

The computational experiments, presented in this paper, are on two classes of random graphs, and are aimed at comparing the different algorithmic building blocks and their impact on average number of steps required to find the MC, as well as the cost per single iteration.

To ensure that hard instances are considered in the test, a preliminary study investigates the empirical hardness as a function of the graph dimension.

3.1. Benchmark graphs

Performance and scalability tests are made on two different classes of random graphs:

Binomial random graphs. A binomial graph $GIL(n,p)$, belonging to Gilbert's model $\mathcal{G}(n,p)$, is constructed by starting from n nodes and adding up to $(n(n-1))/2$ undirected edges, independently with probability $0<p<1$. See [2] for generation details.
Preferential attachment model. A graph instance $PAT(n,d)$, of the preferential attachment model, introduced in [1], is built by starting from a single node and adding successively the remaining nodes. The edges of the newly added nodes are connected to d existing nodes, with preferential attachment to nodes having a higher degree, i.e., with probability proportional to the number of edges present between the existing nodes.

In binomial graphs, the degree distribution for the different nodes will be peaked on the average value, while in the preferential attachment model the probability that a node is connected to k other nodes decreases following a power-law i.e., $P(k) \sim k^{-\gamma}$ with $\gamma>1$.

In the experiments, the graphs are generated using the NetworkX library [10], for a number of nodes ranging from 100 to 1500. Because of the hardness of MC, the optimal solutions of large instances cannot

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

Table 1
Best empirical maximum cliques in the benchmark graphs.

Nodes	*GIL* $(n, 0.3)$	*PAT* $(n, n/3)$
100	6	13
200	7	19
300	8	25
400	8	31
500	8	37
600	8	42
700	9	48
800	9	54
900	9	57
1000	9	60
1100	10	64
1200	10	70
1300	10	74
1400	10	79
1500	10	86

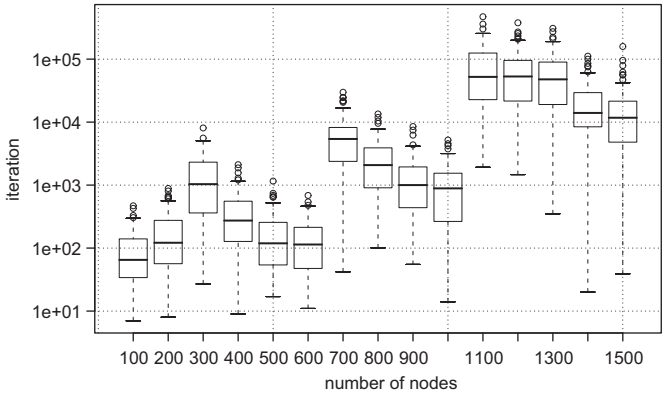

Fig. 4. Iterations of EXPPLAT-RAND to find the *empirical maximum* clique in $GIL(n, 0.3)$. *Y* axis is logarithmic.

be computed and one must resort to the *empirical maximum*. The *empirical maximum* considered in the experiments is the best clique that RLS is able to find in 10 runs of 5 million steps each. In no case DLS-MC with *pd* equal to 1 is able to find bigger cliques for the same number of iterations. The sizes of the empirical MC in the various graphs are listed in Table 1.

The algorithms are tested against our data set, to compute the distribution function of the iterations needed to find the *empirical maximum*. The maximum number of steps per iteration is set to 10 million and each test is repeated on the same graph instance 100 times. For the algorithms having a plateau phase, *maxPlateauSteps* is set to 100.

We count as one iteration each add- or drop-move executed on the clique. DLS-MC code was modified to count the steps in this manner, to be able to make comparisons with the other algorithms. The CPU time spent by each iteration is measured on our reference machine, having one Xeon processor at 3.4 GHz and 6 GB RAM. The operating system is a Debian GNU/Linux 3.0 with kernel 2.6.15-26-686-smp. All the algorithms are compiled with g++ compiler with "-O3 -mcpu = pentium4".

Figs. 4 and 5 summarize with standard box-and-whisker plots the medians, the quartiles, and the outliers of the iterations by EXPPLAT-RAND. Fig. 4 shows that there are some instances which are significantly harder than others. The sawtooth trend of the plot is due to the fact that EXPPLAT-RAND needs on average more iterations to solve instances of the Gilbert model corresponding to the increase of the expected clique size as listed in Table 1. Instances then become easier when the number of nodes increases and the MC remains the same size while the number of optimal cliques increases. This is also confirmed by all other algorithms considered and explained by the theoretical results by Matula [12].

The sawtooth behavior is hardly visible as shown in Fig. 5 because of the different granularity of the cliques dimension with respect to the graph sizes considered.

Fig. 6 reports the number of iterations to find the empirical MC in PAT(1100,366) graphs by the most significant techniques considered in this paper. It can be observed that RLS-LTM achieves the best results for this class of graphs. Furthermore, the variation in the number of iterations is smaller, implying a larger robustness of the technique. Additional results and discussions are presented in Section 3.3.

3.2. Implementation details and cost per iteration of RLS

The total computational cost for solving a problem is of course the product of the number of iterations times the cost of each iteration. More complex algorithms like RLS risk that the higher cost per iteration is not compensated by a sufficient reduction in the total number of iterations. This section is dedicated to exploring this issue.

The original implementation [6] focussed on the algorithm and the appropriate data structures but did not optimize low-level implementation details. For example, every time a new configuration had to be inserted in the table, the memory needed to store the element was allocated dynamically. The new implementation moved from these on-demand allocations to the more efficient allocation of a single bigger chunk of memory; the memory is used as a pool of available locations to be assigned to the elements when needed.

Moreover, the hash table containing the configurations resolves key collisions by means of chaining. In order to keep the frequent access operation as close to a direct access as possible, these chains have to be kept as short as possible. This has been achieved by doubling the size of the hash table when the number of elements inside the table exceeds a specified load factor.

The speedup results are reported in Table 2. They show the improvement in the steps per seconds achieved by the new version, for two random graphs and some representative DIMACS instances. The number of steps per second is computed by measuring on every instance the CPU time spent by the two algorithms to perform 1,000,000 iterations. Let us note that the obtained speedup is substantial. For example, the improvement for large random graphs increases with the graph dimension reaching a factor of 22 for graphs with thousand nodes (C4000.5).

Let us now consider a simple model to capture the time spent by the RLS algorithm on each iteration. Most of the cost is spent on updating the data structures after each addition or deletion. After a node deletion the complexity for updating the data structures is $O(deg_{\overline{G}}(v))$, $deg_{\overline{G}}(v)$ being the degree of the just moved node v in the *complementary* graph $\overline{G}$. After a node addition the complexity is $O(deg_{\overline{G}}(v) \cdot |\text{POSSIBLEADD}|)$, see [6] for more details. Now, because the algorithm alternates between expansions and plateau moves, for most of the run |POSSIBLEADD| oscillates between 0 and 1. We can therefore make the strong assumption that |POSSIBLEADD| is substituted with a small constant. In both cases the dominant factor is therefore $O(deg_{\overline{G}}(v))$.

The computational complexity for using the history data structure can be amortized to a $O(1)$ complexity per iteration. The restart operation cannot be amortized: its complexity is $O(n)$ but it is not performed regularly. On the contrary, the number of restarts highly depends on the search dynamics and on the hardness of the instance.

Under the above assumption we decided to propose an empirical model for the time per iteration which is linear in the number of node and the degree:

$$T(n, deg_{\overline{G}}) = \alpha n + \beta deg_{\overline{G}} + \gamma \quad (1)$$

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

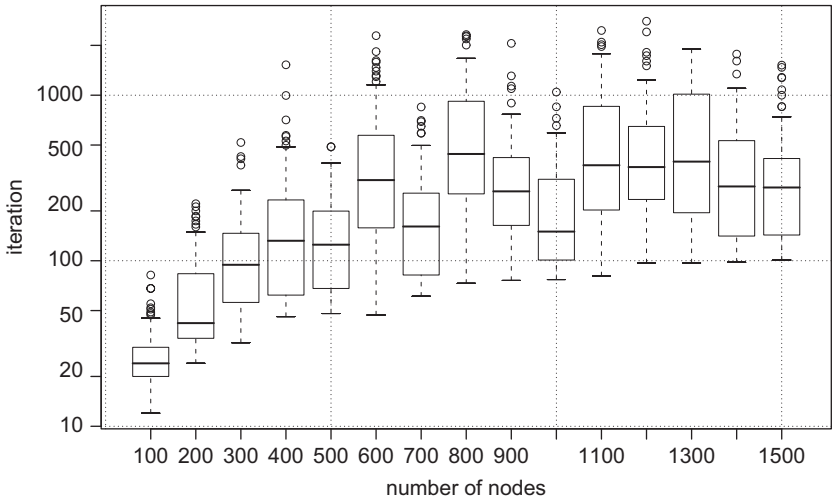

Fig. 5. Iterations of EXPPLAT-RAND to find the *empirical maximum* clique in $PAT(n, n/3)$. Y axis is logarithmic.

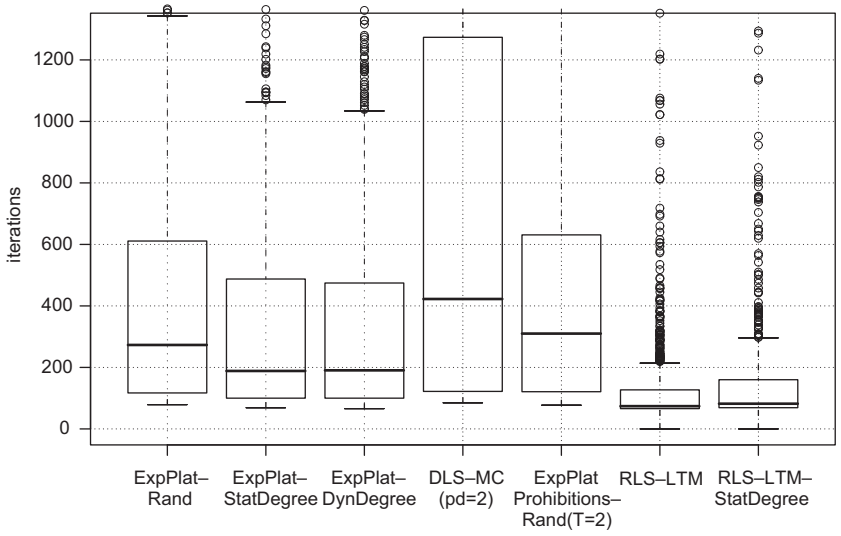

Fig. 6. Iterations to find the *empirical maximum* clique in $PAT(1100, 366)$. Results for the most significant algorithms are reported.

The last simplification is given by substituting the *average* node degree instead of the actual degree.

Let us note that the above model is not precise if the size of the POSSIBLEADD set remains large for a sizable fraction of the iterations. For example, this is the case when a large graph is extremely dense, and the clique is very large. In this case the size of the POSSIBLEADD set is a non-negligible factor which multiplies $deg_{\overline{G}}(v)$, impacting significantly the overall algorithm performance. This happens for the MANN instances in the DIMACS benchmark set which are not considered when fitting the above model.

The fitted model for our specific testing machine is the following:

$$T(n, deg_{\overline{G}}) = 0.0010n + 0.0107deg_{\overline{G}} + 0.0494 \quad (2)$$

The fit residual standard errors for α, β and γ are 0.0004, 0.0009 and 0.2765 respectively.

Let us note that the cost for using the history data structure, which is approximately included in the constant term in the above expression, becomes rapidly negligible as soon as the graph dimension and density are not very small. In fact the memory access costs approximately less than 50 ns per iteration while the total cost reaches rapidly tens of microseconds in the above instances.

3.3. Results summary

Table 3 presents the results on two specific instances of random graphs: $GIL(1100, 0.3)$ and $PAT(1100, 366)$. The choice of $GIL(1100, 0.3)$ is determined by the fact that it is empirically the most difficult instance of our data set, while $PAT(1100, 366)$ is chosen with the same number of nodes. The results are for 100 runs on 10 different instances.

From Table 3 , it is clear that algorithms based only on expansions are not always able to find the MC in the given iteration bound, especially on *hard instances*. The plateau phase increases dramatically the success rate.

Degree consideration is effective in the preferential attachment model, while in Gilbert's graphs, where the nodes tend to have similar degrees, penalty- or prohibition-based algorithms win. For example, the reduction in iterations achieved by EXPPLAT-STATDEGREE over EXPPLAT-RAND on $PAT(1100, 366)$ is

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

Table 2
Speed improvement on random graphs and selected DIMACS benchmark instances of the new RLS implementation.

Instance	Steps per second		Speedup
	RLS [6]	RLS-LTM	
gilbert_1100_0.3	11 202	107 527	9.6
pa_1100_366	24 839	168 350	6.8
C125.9	371 747	1 162 791	3.1
C250.9	281 690	943 396	3.3
C500.9	165 289	714 286	4.3
C1000.9	80 451	471 698	5.9
C2000.9	27 285	265 957	9.7
DSJC500_5	43 290	295 858	6.8
DSJC1000_5	17 422	160 000	9.2
C2000.5	5573	78 125	14.0
C4000.5	1537	34 965	22.8
MANN_a27	485 437	909 091	1.9
MANN_a45	293 255	425 532	1.5
MANN_a81	14 286	16 667	1.2
brock200_2	109 769	543 478	5.0
brock200_4	147 493	699 301	4.7
brock400_2	103 413	555 556	5.4
brock400_4	105 374	552 486	5.2
brock800_2	33 715	264 550	7.8
brock800_4	33 311	262 467	7.9
gen200_p0.9_44	321 543	1 000 000	3.1
gen200_p0.9_55	273 224	943 396	3.5
gen400_p0.9_55	210 084	800 000	3.8
gen400_p0.9_65	204 499	740 741	3.6
gen400_p0.9_75	205 761	724 638	3.5
hamming10-4	46 339	316 456	6.8
hamming8-4	113 122	568 182	5.0
keller4	140 647	546 448	3.9
keller5	55 036	296 736	5.4
keller6	7011	101 626	14.5
p_hat300-1	57 870	308 642	5.3
p_hat300-2	112 233	558 659	5.0
p_hat300-3	171 821	729 927	4.2
p_hat700-1	21 758	168 350	7.7
p_hat700-2	49 358	337 838	6.8
p_hat700-3	88 417	478 469	5.4
p_hat1500-1	7345	85 470	11.6
p_hat1500-2	13 504	184 843	13.7
p_hat1500-3	30 460	282 486	9.3

Some round figures are due to the internal clock resolution.

about 31%. The results are confirmed by a Mann–Whitney U-test (Wilcoxon rank-sum test) at significance level 0.05: p-value is 3.768×10^{-15}. On the contrary, algorithms using degree information have poorer performances on Gilbert's graphs, if compared with their completely random counterparts. For example, EXPPLAT-STATDEGREE finds the MC in $GIL(1100, 0.3)$ only in the 60% of the runs.

The cost per iteration changes significantly among different instances and it also depends on the directions taken in the search-space by the algorithms so that simple model like that derived for RLS is not applicable. For example, the plateau phase does not only decrease the average number of iterations needed to find the MC, but also decrease the time spent by each single iteration. With a plateau phase, in fact, the less frequent restarts have a reduced impact on the average cost per iteration.

Table 3 shows that EXPPLAT-DYNDEGREE spends less time per iteration (factor of 1.4) than EXP-DYNDEGREE in $PAT(1100, 366)$. The results is confirmed by a Mann–Whitney U-test at significance level 0.05: p-value is 1.593×10^{-4}. The improvement is even bigger in $GIL(1100, 0.3)$ where degree-based selections are less appropriate. In this case the factor is 3.8 and is also confirmed by a Mann–Whitney U-test at significance level 0.05: p-value is 1.649×10^{-4}.

In case of dynamic degree selection, the incremental update routine complexity depends also on the size of the POSSIBLEADD set. With plateau phases the search is longer and the POSSIBLEADD set is on average smaller.

RLS, which has a different and less frequent restart policy, alternates between short expansions and plateaus. Therefore the POSSIBLEADD set remains on average smaller than in EXP-DYNDEGREE or EXPPLAT-DYNDEGREE and the cost per iteration is smaller.

Fig. 7 shows the average CPU time per iteration on Gilbert's graphs in log–log scale. The regression lines have a slope of 2.41, 0.92 and 0.95 for EXP-DYNDEGREE RLS and DLS-MC ($pd=2$), respectively, confirming an approximate cost per iteration of EXPPLAT-DYNDEGREE growing faster than n^2, while RLS cost, even if the candidate selection is based on the dynamic degree, grows approximately linearly.

The values about the CPU time to reach the estimated optimal solution listed in Table 3, the CPU time needed on average by RLS-LTM-STATDEGREE to find the MC in Gilbert's hard instance is 88% of the time required by DLS-MC ($pd = 4$). RLS-STATDEGREE needs 330 ms while DLS-MC 374 ms. This is confirmed by a Mann–Whitney U-test at significance level 0.05: p-value is 7.365×10^{-5}.

For the sake of completeness, Appendix A reports the comparison between RLS and DLS-MC on the DIMACS benchmark instances.

3.4. Penalties versus prohibitions

As shown in Table 3, DLS-MC is not always able to find the best clique on *PAT* graphs while prohibition-based heuristic is always successful. Our results confirm that the penalty heuristic tends to be less robust than the prohibition-based heuristic. A significant dependency between DLS-MC performance and the choice of the *pd* parameter is also discussed in [14]. Further investigations, summarized in Fig. 8, show the success rate of DLS-MC compared with that of EXPPLATPROHIBITION-RAND for different values of the *pd* and *prohibition time* parameters. The tests are on all instances of the *PAT* graphs of our data set.

EXPPLATPROHIBITION-RAND is always able to find the MC within 100,000 iterations, while DLS-MC fails for several *pd* values even incrementing the maximum number of iterations by a factor of 10 or 100.

4. Conclusions

The results of the investigation show that a careful implementation of the data structures considering also operating system services like memory allocation achieves a significant reduction of the CPU time per iteration. The implementation of the supporting data structures of the new version has many improvements: (i) the management of the dynamic memory, used for storing the configuration fingerprints in the history, which is not allocated when needed as in the first version, but rather pre-initialized and shared among the steps of the actual run; (ii) the usage of a dynamic hash table where the size is adapted to the load factor; (iii) the substitution of all dynamic allocations in the functions with allocations executed at the beginning and reused throughout the run. Furthermore some algorithmic improvements to the original RLS have been introduced leading to the final RLS-LTM proposal, including algorithmic and implementation changes. RLS-LTM achieves an order of magnitude difference in CPU times for graphs of reasonable sizes, and the difference appears to grow with the problem dimension. This results drastically changes the overall competitiveness of the RLS technique.

The results of the tests on the two graph classes show clearly that the plateau search is necessary to find the MC in hard instances and in

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

Table 3
Results summary with the medians of the empirical steps distribution, the average time per iteration and the total CPU time to reach a solution when it is reached in all tests.

Algorithm	*GIL* (1100, 0.3)			*PAT* (1100, 366)		
	Iter.	μs/*Iter.*	CPU (s)	Iter.	μs/*Iter.*	CPU (s)
EXP-RAND	[92%][a]	8.90	–	[0%][a]	3.20	–
EXP-STATDEGREE	[0%][a]	8.30	–	[40%][a]	3.10	–
EXP-DYNDEGREE	[10%][a]	104.00	–	[0%][a]	10.3	–
EXPPLAT-RAND	74697	5.80	.433	273	3.10	.00084
EXPPLAT-STATDEGREE	[60%][a]	5.70	–	189	3.10	.00058
EXPPLAT-DYNDEGREE	75 577	27.20	2.055	191	7.55	.00144
DLS-MC ($pd = 2$)	75 943	5.90	.448	423	3.20	.00135
DLS-MC ($pd = 4$)	63 467	5.90	.374	[99%][a]	3.20	–
DLS-MC ($pd = 8$)	73 831	5.90	.453	[85%][a]	3.20	–
EXPPLATPRO.-RAND ($T = 2$)	65 994	5.80	.382	310	3.10	.00096
EXPPLATPRO.-RAND ($T = 4$)	67 082	5.90	.395	333	3.10	.00103
EXPPLATPRO.-RAND ($T = 8$)	67 329	5.80	.390	329	3.15	.00103
RLS	47 526	90.22	4.287	222	37.27	.00827
RLS-LTM	47 442	9.40	.445	75	5.50	.00041
RLS-LTM-STATDEGREE	45 259	7.30	.330	84	4.50	.00037

[a]The algorithm is not always able to find the maximum clique; the percent of successes is reported in these cases.

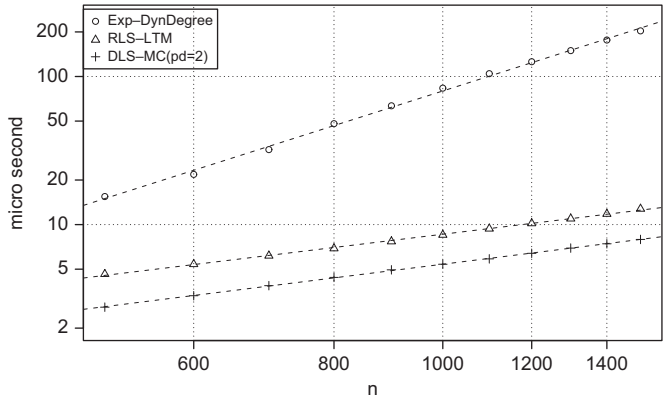

Fig. 7. Empirical cost per iteration in μs on Gilbert's graphs. Log–log scale.

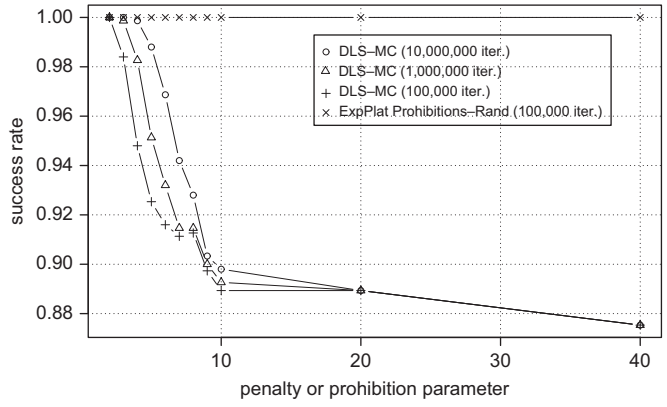

Fig. 8. Success ratio of penalty- and prohibition-based algorithms on instances of the preferential attachment model.

any case to reduce the average number of iterations. The complexity added to the algorithms by the plateau search does not increase the cost per iteration. On the contrary, especially for the algorithms using the dynamic degree for candidate selections, it reduces the CPU time per iteration.

On Gilbert's graphs, where the nodes have the same degree on average, prohibition- or penalty-based algorithms perform better than pure random selections. On instances of the preferential attachment model, algorithms selecting the nodes using information about the degree are faster.

On the contrary, degree-based algorithms have poorer performance than random-selection algorithms in Gilbert's graphs, while prohibition- and penalty-based algorithms are disadvantageous in the preferential attachment model. The penalty heuristic is less robust than the prohibition heuristic, depending on the appropriate selection of the penalty value.

RLS-LTM and RLS-LTM-STATDEGREE always perform better than the other algorithms. The cost per iteration of RLS-LTM-STATDEGREE is bigger than the one of DLS-MC, although of the same order of magnitude. But the fewer steps needed on average to find the best cliques make RLS-LTM the best choice for the two graph models considered in this paper, especially considering that no detailed tuning is executed in RLS before running the comparison.

The software corresponding to the algorithm, benchmark graphs and the heuristically optimal values are available at request for research purposes.

Acknowledgments

We thank Holger Hoos and the co-authors for making available the software corresponding to the DLS-MC algorithm.

Appendix A. DIMACS benchmark set

The following table compares DLS-MC (PD = OPT) and RLS on a selected "snapshot" of the DIMACS benchmark set. The results presented are averages on 100 runs of 100,000,000 maximum steps each. DLS-MC (PD = OPT) is DLS-MC with the *pd* parameter set to the optimal value for each single instance as suggested in [14].

For each instance and both algorithms Table 4 shows the median clique size with the median percentage deviation from the best known. The CPU time and iteration medians and interquartile ranges (IQRs) are reported only for successful runs.

One algorithm dominates the other if it has a bigger median solution quality, or a smaller median percentage deviation from the best known. If both algorithms are able to find the MC on every run, the

Table 4
Algorithm comparison on a selected sub-set of the DIMACS benchmark instances.

Instance	Best	DLS-MC (pd = opt)			RLS-LTS		
		Solution quality	CPU(s)	Steps	Solution quality	CPU(s)	Steps
C125.9	34	34 (0.00)	<0.001	175 (215)	**34** (**0.00**)	<0.001	**88** (**118**)
C250.9	44	44 (0.00)	<0.001	1348 (1770)	44 (0.00)	<0.001	1060 (1010)
C500.9	57	**57** (**0.00**)	**0.030** (**0.040**)	**59 780** (**76 900**)	57 (0.00)	0.115 (0.253)	82 740 (181 010)
C1000.9	68	**68** (**0.00**)	**0.360** (**0.630**)	**409 500** (**716 300**)	68 (0.00)	1.465 (2.035)	703 000 (973 400)
C2000.9	78	78 (0.00)	–	–	78 (0.00)	–	–
DSJC1000_5	15	15 (0.00)	0.330 (0.527)	81 560 (130 700)	**15** (**0.00**)	**0.200** (**0.335**)	**31 720** (**53 220**)
DSJC500_5	13	13 (0.00)	0.010 (0.010)	2454 (3954)	**13** (**0.00**)	**0.000** (**0.010**)	**1131** (**1692**)
C2000.5	16	16 (0.00)	0.370 (0.743)	59 000 (117 180)	**16** (**0.00**)	**0.465** (**0.640**)	**35 760** (**49 150**)
C4000.5	18	18 (0.00)	119.000 (143.920)	9 686 000 (11 710 000)	**18** (**0.00**)	**107.000** (**147.820**)	**3 705 000** (**5 113 000**)
MANN_a27	126	**126** (**0.00**)	**0.020** (**0.010**)	**60 240** (**26 820**)	126 (0.00)	0.070 (0.073)	62 760 (72 060)
MANN_a45	345	344 (0.29)	–	–	344 (0.29)	–	–
MANN_a81	1099	1 098 (0.09)	–	–	1098 (0.09)	–	–
brock200_2	12	**12** (**0.00**)	**0.010** (**0.020**)	**12 230** (**20 680**)	12 (0.00)	0.090 (0.150)	49 100 (83 670)
brock200_4	17	**17** (**0.00**)	**0.030** (**0.040**)	**39 960** (**51 020**)	17 (0.00)	0.190 (0.493)	135 000 (346 480)
brock400_2	29	**29** (**0.00**)	**0.230** (**0.362**)	**232 800** (**371 300**)	29 (0.00)	–	–
brock400_4	33	**33** (**0.00**)	**0.030** (**0.040**)	**28 770** (**37 720**)	33 (0.00)	3.365 (4.530)	1 890 000 (2 521 300)
brock800_2	24	**24** (**0.00**)	**7.870** (**10.178**)	**3 397 000** (**4 387 000**)	21 (14.29)	–	–
brock800_4	26	**26** (**0.00**)	**3.275** (**4.442**)	**1 410 000** (**1 914 600**)	21 (23.81)	–	–
gen200_p0.9_44	44	44 (0.00)	<0.001	1 934 (3981)	44 (0.00)	<0.001	1535 (1832)
gen200_p0.9_55	55	**55** (**0.00**)	<0.001	**333** (**827**)	55 (0.00)	<0.001	596 (562)
gen400_p0.9_55	55	**55** (**0.00**)	**0.010** (**0.020**)	**33 920** (**57 910**)	55 (0.00)	0.030 (0.040)	21 160 (30 150)
gen400_p0.9_65	65	**65** (**0.00**)	<0.001	**1001** (**1480**)	65 (0.00)	<0.001	1294 (1132)
gen400_p0.9_75	75	**75** (**0.00**)	<0.001	**507** (**777**)	75 (0.00)	<0.001	1576 (1184)
hamming8-4	16	16 (0.00)	<0.001	28 (16)	**16** (**0.00**)	<0.001	**16** (**0**)
hamming10-4	40	40 (0.00)	0.000 (0.010)	2469 (3241)	**40** (**0.00**)	**0.000** (**0.010**)	**529** (**1044**)
keller4	11	11 (0.00)	<0.001	40 (44)	**11** (**0.00**)	<0.001	**11** (**8**)
keller5	27	27 (0.00)	0.010 (0.020)	6345 (8798)	**27** (**0.00**)	**0.010** (**0.030**)	**2828** (**7150**)
keller6	59	59 (0.00)	–	–	**59** (**0.00**)	**6.765** (**12.482**)	**686 500** (**1 265 300**)
p_hat300-1	8	8 (0.00)	<0.001	163 (266)	8 (0.00)	<0.001	128 (192)
p_hat300-2	25	25 (0.00)	<0.001	113 (108)	**25** (**0.00**)	<0.001	**27** (**20**)
p_hat300-3	36	36 (0.00)	<0.001	530 (806)	36 (0.00)	<0.001	633 (1232)
p_hat700-1	11	11 (0.00)	0.010 (0.010)	2014 (3276)	**11** (**0.00**)	**0.010** (**0.020**)	**1336** (**1898**)
p_hat700-2	44	44 (0.00)	<0.001	281 (322)	**44** (**0.00**)	<0.001	**112** (**83**)
p_hat700-3	62	62 (0.00)	<0.001	585 (501)	**62** (**0.00**)	<0.001	**219** (**282**)
p_hat1500-1	12	12 (0.00)	1.330 (1.720)	207 900 (269 230)	**12** (**0.00**)	**1.690** (**2.090**)	**145 400** (**180 920**)
p_hat1500-2	65	**65** (**0.00**)	**0.000** (**0.010**)	**784** (**1061**)	65 (0.00)	0.010 (0.010)	331 (1244)
p_hat1500-3	94	**94** (**0.00**)	**0.000** (**0.010**)	**1867** (**2610**)	94 (0.00)	0.010 (0.010)	1253 (1498)

The table shows the median solution quality and within brackets the median percentage deviation from the best known, as well as CPU seconds and steps medians with IQR within brackets. The dominating algorithm is highlighted in bold.

dominating algorithm is the one having either a smaller median CPU time or, in the case of no measurable difference in the CPU times, a smaller number of iterations. The comparisons are assessed statistically by means of a Mann–Whitney U-test (Wilcoxon rank-sum test) at significance level 0.05. The dominating algorithm is highlighted in bold.

Let us note again that the comparison below is not fair, because in one case (DLS-MC) one reports only the time corresponding to the optimal setting of an individual pd parameter for each instance, while in the second case this extensive tuning phase is absent.

In most cases, apart from the "camouflaged" Brockington–Culberson graphs [7], the optimal values obtained are the same. These graphs have been designed to be difficult for greedy algorithms; therefore it is not surprising that the greedy node selection of RLS negatively impacts on the performance on those graphs. For the CPU times, in many cases the graph dimension is so small that the measure becomes difficult; in some other cases RLS has times which are larger but of the same order of magnitude. For other instances RLS CPU time is shorter, which is quite unexpected given the absence of the tuning phase.

References

[1] Barabasi AL, Albert R. Emergence of scaling in random networks. Science 1999;286:509–12.
[2] Batagelj V, Brandes U. Efficient generation of large random networks. Physical Review E 2005;71(3):036113.
[3] Battiti R, Protasi M. Reactive local search for the maximum clique problem. Technical Report TR-95-052, ICSI, 1947 Center St. – Suite 600 – Berkeley, California, September 1995.
[4] Battiti R, Tecchiolli G. The reactive tabu search. ORSA Journal on Computing 1994;6(2):126–40.
[5] Battiti R, Bertossi AA. Greedy, prohibition, and reactive heuristics for graph partitioning. IEEE Transactions on Computers 1999;48(4):361–85.
[6] Battiti R, Protasi M. Reactive local search for the maximum clique problem. Algorithmica 2001;29(4):610–37.
[7] Brockington M, Culberson JC. Camouflaging independent sets in quasi-random graphs. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability: second DIMACS implementation challenge, vol. 26, American Mathematical Society; 1996. p. 75–88.
[8] Butenko S, Wilhelm W. Clique-detection models in computational biochemistry and genomics. Journal of Operational Research 2006;173:1–17.
[9] Håstad J. Clique is hard to approximate within $n^{1-\varepsilon}$. In: Proceedings of 37th annual IEEE symposium on foundations of computer science, IEEE Computer Society; 1996. p. 627–36.
[10] Hagberg A, Schult D, Swart P. NetworkX library developed at the Los Alamos National Laboratory Labs Library (DOE) by the University of California, 2004. Code available at: ⟨https://networkx.lanl.gov/⟩.

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013

[11] Ji Y, Xu X, Stormo GD. A graph theoretical approach to predict common RNA secondary structure motifs including pseudoknots in unaligned sequences. Bioinformatics 2004;20(10):1591–602.
[12] Matula DW. The largest clique size in a random graph. Southern Methodist University; 1976.
[13] Pardalos PM, Xue J. The maximum clique problem. Journal of Global Optimization 1994;4:301–28.
[14] Pullan W, Hoos HH. Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research 2006;25:159–85.
[15] Battiti R, Brunato M, Mascia F. Reactive search and intelligent optimization operations research/computer science interfaces series, vol. 45. Springer, November 2008.

Please cite this article as: Battiti R, Mascia F. Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research (2009), doi:10.1016/j.cor.2009.02.013