Contributions to Discrete Mathematics

Volume 13, Number 2, Pages 101–111
ISSN 1715-0868

SMALL ON-LINE RAMSEY NUMBERS — A NEW APPROACH

PRZEMYSŁAW GORDINOWICZ AND PAWEŁ PRAŁAT

Abstract. In this note, we revisit the problem of calculating small on-line Ramsey numbers $\overline{\mathcal{R}}(G,H)$. A new approach is proposed that reduces the running time of the algorithm determining that $\overline{\mathcal{R}}(K_3,K_4)=17$ by a factor of at least $2\cdot 10^6$ compared to the previously used approach. Using high performance computing networks, we determined that $\overline{\mathcal{R}}(K_4,K_4)\geq 26$, $\overline{\mathcal{R}}(K_3,K_5)\geq 25$, and that $\overline{\mathcal{R}}(K_3,K_3,K_3)\geq 20$ for a natural generalization to three colours. All graphs on 3 or 4 vertices are investigated as well, including nonsymmetric cases.

1. Introduction and definitions

In this paper, we consider on-line Ramsey numbers introduced in 2005 by Kurek and Ruciński [8] and the corresponding on-line Ramsey game. (The game was considered earlier by Beck [1] but not in terms of the numbers; Friedgut et al. [5] also studied a variant of this game but in the context of the random graph theory.) Let G and H be two fixed graphs. The game between two players, called Builder and Painter, is played on an unbounded set of vertices. In each of her moves, Builder draws a new edge which is immediately coloured red or blue by Painter. The goal of Builder is to force Painter to create a red copy of a graph G or a blue copy of a graph H; the goal of Painter is the opposite, he is trying to avoid it for as long as possible. The payoff to Painter is the number of moves until this happens and he seeks the highest possible payoff. Since this is a two-person, full information game with no ties, one of the players must have a winning strategy. *The on-line Ramsey number* $\overline{\mathcal{R}}(G,H)$ is the smallest payoff over all possible strategies of Builder, assuming Painter uses an optimal strategy. Note that this number is well defined since Builder may present edges of the Ramsey graph for G and H.

Received by the editors December 30, 2016, and in revised form October 10, 2017.

2000 *Mathematics Subject Classification.* 91A46.

Key words and phrases. on-line Ramsey number, size Ramsey number.

The second author is supported in part by NSERC, Ryerson University, and SHARCNET.

Similar to the classical Ramsey numbers (see a dynamic survey of Radziszowski [15] which includes all known nontrivial values and bounds for Ramsey numbers), it is difficult to compute the exact value of $\overline{\mathcal{R}}(G,H)$ unless G,H are trivial. In the area of small on-line Ramsey numbers, very little is known. For example, by dealing with many cases, Grytczuk et al. [7] determined on-line Ramsey numbers for short paths on 6 or less vertices. The second author of this paper, with computer support, extended it to paths on 9 or less vertices (see [12, 14] for more details).

In [8], Kurek and Ruciński considered the most interesting case where G and H are cliques, but besides the trivial $\overline{\mathcal{R}}(K_2,K_k)=\binom{k}{2}$, they were able to determine only one more value, namely $\overline{\mathcal{R}}(K_3,K_3)=8$. Only one more value is known up to date, namely, $\overline{\mathcal{R}}(K_3,K_4)=17$. It was obtained, with computer support, by the second author in [13]. The total computational requirements were estimated to be around $1,379,000$ CPU hours! In fact, 17 matches a trivial upper bound for $\overline{\mathcal{R}}(K_3,K_4)$ that can be obtained by mimicking the proof of the upper bound for classical Ramsey number $R(K_3,K_4)$. This observation can be generalized and it follows that for all $2\le k\le \ell$

$$\overline{\mathcal{R}}(K_k,K_\ell)\le \frac{3}{2}\sum_{i=0}^{k-1}\binom{2i}{i}+\binom{k+\ell-1}{\ell-1}-\binom{2k-1}{k-1}-\ell-k+\frac{1}{2}\,, \tag{1.1}$$

which gives an asymptotic upper bound of $\frac{3}{8\sqrt{\pi}}\frac{4^k}{\sqrt{k}}$ for diagonal numbers $\overline{\mathcal{R}}(K_k,K_k)$. (See [13] for more details.) In a table below, we present small values of this trivial upper bound of $\overline{\mathcal{R}}(K_k,K_\ell)$ for $3\le k\le \ell\le 10$.

	3	4	5	6	7	8	9	10
3	8	17	31	51	78	113	157	211
4		36	70	125	208	327	491	710
5			139	264	473	802	1296	2010
6				515	976	1767	3053	5054
7					1899	3614	6616	11620
8						7045	13479	24918
9							26348	50657
10								99276

TABLE 1. Upper bounds of $\overline{\mathcal{R}}(K_k,K_\ell)$

This note is devoted to small numbers but let us mention the following asymptotic conjecture that seems to be the most important one in this area. It is clear that

$$\overline{\mathcal{R}}(K_k,K_k)\le \binom{R(K_k,K_k)}{2}\,,$$

since Builder can present edges of $K_{R(K_k,K_k)}$ (in any sequence) and Painter cannot avoid a monochromatic copy of K_k. The following intriguing conjecture was posed by Kurek and Ruciński [8].

Conjecture ([8]).

$$\lim_{k\to\infty} \frac{\binom{R(K_k,K_k)}{2}}{\overline{\mathcal{R}}(K_k,K_k)} = \infty$$

The main breakthrough related to this conjecture, due to Conlon [2], is the result that shows that there exists a constant $c > 1$ such that

$$\frac{\binom{R(K_k,K_k)}{2}}{\overline{\mathcal{R}}(K_k,K_k)} \geq c^k$$

for infinitely many values of t. Also in [2], a more specific upper bound for $\overline{\mathcal{R}}(K_k,K_k)$ was provided, showing that there exists a constant $c > 0$ such that

$$\overline{\mathcal{R}}(K_k,K_k) \leq k^{-c\log k/\log\log k}4^k.$$

1.1. **Our results.** In this note, we revisit the algorithm for calculating small on-line Ramsey numbers. In [13], a refinement of the brute force approach was used, which systematically searches for a solution to a problem among all available options. Since it was not possible to examine all possibilities, many validity criteria were used to determine which portion of the solution space needed to be searched. Having said that, the main idea was to handle all possible configurations that can occur during the game after t moves, starting from $t = 1$ and building up. In this note, a new approach is proposed that reduces the running time of the algorithm dramatically. As a warm-up test, we rechecked that $\overline{\mathcal{R}}(K_3,K_4) = 17$ in 30 minutes. Hence, the running time decreased by a factor of more than $2 \cdot 10^6$ compared to the previously used approach! Unfortunately, despite the fact that we made such a good improvement, the new approach is not enough to determine new values but provides only new lower bounds for complete graphs. Using high performance computing networks, we determined the following lower bounds:

$$26 \leq \overline{\mathcal{R}}(K_4,K_4) \leq 36, \tag{1.2}$$

$$25 \leq \overline{\mathcal{R}}(K_3,K_5) \leq 31, \tag{1.3}$$

$$20 \leq \overline{\mathcal{R}}(K_3,K_3,K_3) \leq 24. \tag{1.4}$$

($\overline{\mathcal{R}}(G,H,I)$ is a natural generalization of $\overline{\mathcal{R}}(G,H)$ to three colours.) Computations for $\overline{\mathcal{R}}(K_4,K_4)$ required 600 CPU hours, $\overline{\mathcal{R}}(K_3,K_5)$ required 4,100 CPU hours, and $\overline{\mathcal{R}}(K_3,K_3,K_3)$ required 16,300 CPU hours. We also checked other graphs on 3 or 4 vertices, including nonsymmetric cases. Our results are presented in Table 2. Recall that the paw is a graph on 4 vertices obtained by attaching an edge to a vertex of the triangle; the diamond consists of a complete graph K_4 with one edge removed. (For this and other standard definitions the reader is directed to [16].)

	P_3	K_3	P_4	C_4	$K_{1,3}$	paw	diamond	K_4
P_3	3	5	4	6	4	6	7	10
K_3		8	8	11	8	9	12	17
P_4			5	8	6	8	10	15
C_4				10	8	11	14	≥ 20
$K_{1,3}$					5	8	11	15
paw						10	13	19
diamond							16	≥ 21
K_4								≥ 26

TABLE 2. Values of $\overline{\mathcal{R}}(G, H)$ for connected graphs on 3 or 4 vertices

The program used and more statistics can be downloaded from [6]. The referees checked the mathematical part but not the computer code (that is, the results have not been verified completely by the referees). Let us mention that $\overline{\mathcal{R}}(C_4, C_4) = 10$ was obtained independently in [3] by a direct proof. Short cycles vs. short paths were investigated in [4].

2. Algorithm

We already know that $\overline{\mathcal{R}}(K_3, K_4) = 17$. However, in order to demonstrate the main ideas of the algorithm and to benchmark it, let us suppose that our goal is to reverify this. As already mentioned, in fact, the upper bound of 17 follows immediately from (1.1) so the goal of the algorithm is to provide the lower bound.

2.1. **Brief description.** It is convenient to assume that Builder uses her best strategy. Then, regardless of the strategy used by Painter, after 17 moves there is either a red copy of K_3 or a blue copy of K_4. What can be said about the board one move earlier? It is clear that after 16 moves, provided the game is not over yet, the board contains a red copy of $K_3 \setminus e$ and a blue copy of $K_4 \setminus e$ that are "glued" on a missing edge (see Figure 1; dashed edges correspond to red edges, solid edges to blue edges, and the doted edge corresponds to the missing edge). This configuration is clearly sufficient to finish the game in one more round (Builder simply presents the "missing" edge). On the other hand, if no such configuration is present, then Painter has a safe move and the game will not be finished in the next round. Hence, this configuration is necessary in order to be able to finish the game in one more move. Our goal will be to determine the family of all configurations that guarantees Builder wins in 17 moves. Let $\mathcal{C}_{17}$ be the set consisting of two configurations: a red K_3 and a blue K_4. We generate sets $\mathcal{C}_k$ inductively, starting from $k = 16$ and reaching $k = 1$ at the end of this process. The set $\mathcal{C}_k$ will consist of configurations with the following property: if the board contains some configuration from $\mathcal{C}_k$, then Builder

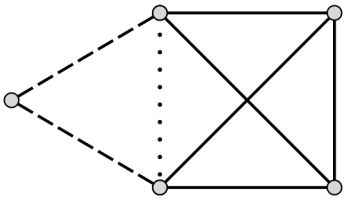
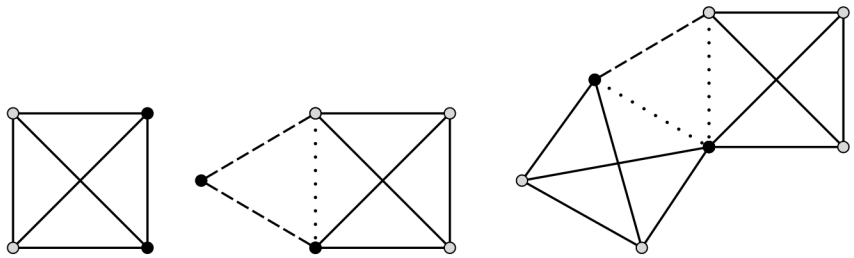


FIGURE 1. Configuration one move before the end of the game

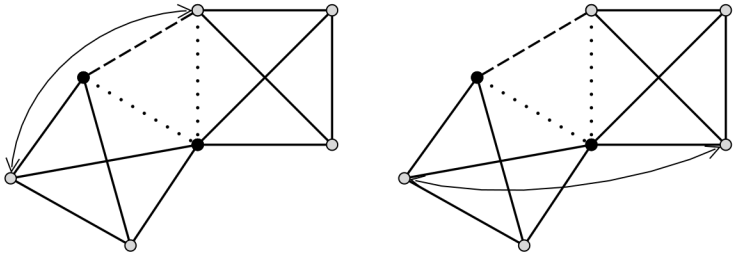
has a strategy to finish the game in at most $17 - k$ rounds. Finally, if $\mathcal{C}_1$ contains the following two configurations, one consisting of a red edge only and another one consisting of a blue edge only, then our goal is achieved. (In the final set C_1, "missing" edges are ignored; in other words, we focus exclusively on coloured edges.) It is obvious that Painter cannot avoid both configurations after Builder presents the first edge and so $\overline{\mathcal{R}}(K_3, K_4) \leq 17$. Moreover, if these two configurations are not both present in any $\mathcal{C}_k$ for some $k \geq 2$, then Builder cannot force Painter to give up earlier and, in fact, $\overline{\mathcal{R}}(K_3, K_4) = 17$.

2.2. **Inductive step.** Formally, a *configuration* $A = (G, c) \in \mathcal{C}_k$, that represents a given situation on the board after k moves, consists of a graph $G = (V, E)$ and a function $c : E(G) \to \{0, 1, 2\}$; $c(e) = 1$ indicates that an edge e is red, $c(e) = 2$ indicates that e is blue, $c(e) = 0$ indicates that e is not yet presented. Other edges (not in $E(G)$) may or may not be present at this point of the game. In order to simplify the notation, we will write $e \in E(A)$ instead of $e \in E(G)$, $v \in V(A)$ instead of $v \in V(G)$, G_A instead of G, and c_A instead of c.

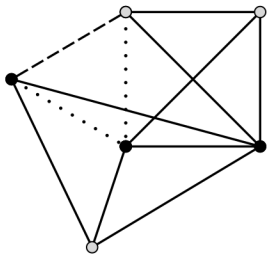
Suppose that all sets $\mathcal{C}_j$ for $j > i$ are already determined and we would like to find all elements of $\mathcal{C}_i$. What do we expect from a configuration $C \in \mathcal{C}_i$? We need to be sure that, starting from C, Builder is able to present an edge so that, regardless of what Painter does, some configuration from $\bigcup_{j>i} \mathcal{C}_j$ is achieved. On the other hand, Painter should be able to achieve some configuration from $\mathcal{C}_{i+1}$; indeed, if this is not possible, then Builder can finish the game earlier and, as a result, C is already present in some $\mathcal{C}_j$ for $j > i$. It follows that it is enough to select two configurations, $A \in \mathcal{C}_{i+1}$ and $B \in \bigcup_{j>i} \mathcal{C}_j$, and then, for every pair of edges $e_a \in E(A)$ and $e_b \in E(B)$ of different colour ($c_A(e_a), c_B(e_b) \in \{1, 2\}$ and $c_A(e_a) \neq c_B(e_b)$), we create a new configuration C that is obtained by removing e_a from A, removing e_b from B, and "gluing" these two configurations by identifying the removed edges (e_a and e_b)—see Figure 2 (a–b). (Note that there are two ways to "glue" the two edges, which may or may not be isomorphic.) When the game reaches configuration C, Builder can simply present an edge $e_a = e_b$ and Painter is forced to go to configuration A or B.



(a) Two configurations with selected edges (b) "Glued" configuration



(c) Illegal merging (d) Legal merging



(e) Result of merging

FIGURE 2. Generating and merging configuration

Note that endpoints of e_a and e_b are perhaps not the only vertices that are identified with each other. However, in order for vertex v_a from A to be identified with vertex v_b from B it is required that no edge in C corresponds to edges in A and B that are coloured differently. See Figure 2 for an example: on Figure 2 (b) only endpoints of e_a, e_b are identified, Figure 2 (c) has one more pair identified that is illegal, Figure 2 (d) presents a legal merging while Figure 2 (e) shows a configuration that is obtained as a result of this merging. Formally, for two configurations A, B, and two edges $e_a = \{v_a, v'_a\} \in E(A)$, $e_b = \{v_b, v'_b\} \in E(B)$ of different colour (that is, either $c_A(e_a) = 1$ and $c_B(e_b) = 2$, or $c_A(e_a) = 2$ and $c_B(e_b) = 1$), we search for *all* minimal configurations C such that there are two one-to-one functions $f_a : V(A) \to V(C)$ and $f_b : V(B) \to V(C)$ such that

- $f_a(v_a) = f_b(v_b)$ and $f_a(v'_a) = f_b(v'_b)$ (or $f_a(v_a) = f_b(v'_b)$ and $f_a(v'_a) = f_b(v_b)$),
- $c_C(\{f_a(v_a), f_a(v'_a)\}) = 0$,
- for every $u, v \in V(A)$, if $\{u, v\} \neq e_a$ and $c_A(\{u, v\}) > 0$, then $c_A(\{u, v\}) = c_C(\{f_a(u), f_a(v)\})$,
- for every $u, v \in V(B)$, if $\{u, v\} \neq e_b$ and $c_B(\{u, v\}) > 0$, then $c_B(\{u, v\}) = c_C(\{f_b(u), f_b(v)\})$.

Moreover, it is obvious that any configuration $C \in \mathcal{C}_k$ has no more than k coloured edges, as only such configurations can be achieved after k moves. Formally, we would like to make sure that $m^1_{V(C)V(C)} + m^2_{V(C)V(C)} \leq k$ for any configuration $C \in \mathcal{C}_k$, where m^c_{XY} denotes the number of edges between vertices from sets X and Y that are coloured in colour c, that is,

$$m^c_{XY} = |\{\{x, y\} \colon x \in X \wedge y \in Y \wedge x \neq y \wedge c_C(\{x, y\}) = c\}|.$$

2.3. **The algorithm.** We get the following algorithm for verifying whether $\overline{\mathcal{R}}(G, H)$ is at most some value of k—see procedure `Ramsey` below.

Procedure `Ramsey` (*Is* $\overline{\mathcal{R}}(G, H) \leq k$*?*).

Input: $\mathcal{C}_k$ consists of two configurations, a red copy of G and a blue copy of H.
Output: 1 if $\overline{\mathcal{R}}(G, H) \leq k$ and 0 otherwise.

```
1:  for i := k − 1 downto 1 do
2:      for each configuration A ∈ C_{i+1} do
3:          for each configuration B ∈ ⋃_{j>i} C_j do
4:              for each pair of edges e_a ∈ E(A) and e_b ∈ E(B) of different
                colour do
5:                  create new configurations C_1, C_2 identifying endpoints of
                    e_a and e_b
6:                  for each configuration D obtained after merging vertices
                    in C_x do
7:                      if D contains no more than i coloured edges then
8:                          put D into C_i
9:  if C_1 contains a configuration consisting of a red edge only and a con-
    figuration consisting of a blue edge only then
10:     return 1
11: else
12:     return 0
```

Clearly, the algorithm creates many configurations that are identical, up to isomorphism. To remove unnecessary configurations, we use Brendan McKay's `nauty` software package [9] for computing automorphism groups of graphs and digraphs (see [10, 11] for more details). Moreover, since the game is played on an infinite board, a vertex with no coloured edge adjacent to it carries no constraint for either player and can be removed from a

configuration. An algorithm in this form took roughly 3,000 CPU hours to verify that $\overline{\mathcal{R}}(K_3, K_4) > 16$ (more than 450 times faster compared to the previously used approach).

2.4. **Improvements.** Below, we discuss a few crucial improvements of this algorithm. We first describe a part that generates all minimal configurations C. Suppose that for some $i \geq 1$, $A \in \mathcal{C}_{i+1}$, $B \in \bigcup_{j>i} \mathcal{C}_j$, and two edges $e_a = \{v_a, v'_a\} \in E(A)$, $e_b = \{v_b, v'_b\} \in E(B)$ of different colour are given. Every configuration C will have at most $|V(A)| + |V(B)| - 2$ vertices so we may start with an empty graph ($E(C) = \emptyset$) on $|V(A)| + |V(B)| - 2$ vertices and a trivial function c_C. Once a configuration is generated and is ready to be put in $\mathcal{C}_i$, we simply remove all isolated vertices that are irrelevant. We fix $f_a : V(A) \to V(C)$, any one-to-one function, and update $E(C)$, c_C so that the desired conditions for configuration A are satisfied (that is, configuration C consists now of a copy of a graph from A and $|V(B)| - 2$ isolated vertices). Our goal is to generate all possible one-to-one functions $f_b : V(B) \to V(C)$, starting with assigning $f_b(v_b) = f_a(v_a)$ and $f_b(v'_b) = f_a(v'_a)$ (or, in an independent instance, $f_b(v_b) = f_a(v'_a)$ and $f_b(v'_b) = f_a(v_a)$), such that the desired conditions are satisfied. Each discovered function will yield a minimal configuration C so we may focus on the process of generating f_b.

A trivial approach would be to try all functions f_b, assigning vertices of $V(C)$ to $V(B)$ one by one. Every time a new vertex is assigned, we verify conditions on the already assigned part (which usually requires updating $E(C)$ and c_C) and cut the branch if at least one condition fails. An important improvement that we introduced is to try to predict which branches are going to create more than i coloured edges (before it actually happens!) and cut these branches in advance so that this fraction of the solution space does not need to be searched. We are allowed to do that, since no configuration from these branches will be added to $\mathcal{C}_i$ anyway. We use our knowledge about the configuration B, and for each vertex in $V(C)$, we keep an information about the number of adjacent coloured edges already assigned to them and the number of coloured edges that are adjacent to vertices of B that wait to be assigned. Many ad-hoc arguments as well as some variants of the well-known Hungarian method are used to get a good estimation for the minimum number of edges that will be present in C but are not assigned yet. (The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time.) An algorithm in this form took roughly 15 CPU hours to verify that $\overline{\mathcal{R}}(K_3, K_4) > 16$ (an improvement by a factor of more than 200).

More formally, suppose that at some step of the algorithm we have assigned vertices of some subset $X \subseteq V(B)$; let $Y = f_b(X)$ and $Z = f_a^{-1}(Y)$. Any final configuration C will satisfy for $c \in \{1, 2\}$

$$m^c_{V(C)V(C)} \geq m^c_{YY} + s^c + t^c,$$

where t^c and s^c are *any* lower bounds for the number of edges between X and its complement, and within the complement of X, respectively (in any final configuration). Clearly, we are interested in getting as large values of t^c and s^c as possible but trivially we get

$$\begin{aligned} s^c &\geq \max\left\{m^c_{(V(A)\setminus Z)(V(A)\setminus Z)}, m^c_{(V(B)\setminus X)(V(B)\setminus X)}\right\}, \\ t^c &\geq \sum_{y\in Y} \max\left\{m^c_{\{f_a^{-1}(y)\}(V(A)\setminus Z)}, m^c_{\{f_b^{-1}(y)\}(V(B)\setminus X)}\right\}. \end{aligned}$$

The above bounds can be calculated efficiently. Moreover, the bound for t^c can be improved using a more sophisticated (optimal assignment) approach. For $c \in \{1,2\}$, each element $u \in V(C)\setminus Y$ and each $v \in V(B)\setminus X$ let

$$t^c_{uv} = \sum_{y\in Y} \max\left\{m^c_{\{y\}\{u\}}, m^c_{\{f_b^{-1}(y)\},\{v\}}\right\}.$$

Note, that any extension of function f_b which maps the whole configuration B into C will produce at least t^c c-coloured edges where t^c is the optimal transversal of the matrix $T^c = [t^c_{uv}]$ (if needed, we may add isolated vertices to $V(B)$ to make sure T^c is a square matrix). Of course, it may be not possible as optimal assignment of the vertices $V(B)$ to $V(C)$ may produce some colour conflict, but this bound can be calculated faster. In the algorithm we mix the above two approaches with some heuristic decision whether to compute value of t^c precisely or just use the quick bound.

The next improvement uses a straightforward observation but is an important one. When two edges $e_a = \{v_a, v'_a\} \in E(A)$ and $e_b = \{v_b, v'_b\} \in E(B)$ of different colours are processed, it is desirable to test only nonisomorphic pairs of configurations with an edge selected. Of course, one can simply test all pairs and remove isomorphic copies once $\mathcal{C}_i$ is generated but discovering isomorphic cases in advance saves a lot of time. This improvement can be easily done using the `nauty` software package one more time. For example, for every red edge in configuration A, we create a copy of this configuration with the selected edge recoloured to, say, light red, and then we remove all unnecessary (isomorphic) configurations. We do the same for every blue edge in configuration B and now we are ready for "gluing" configurations on "light" edges, dealing with important pairs of edges only.

Let us also mention the following improvement. It is clear that a configuration $A \in \mathcal{C}_i$ can be safely removed if there exists a subconfiguration $B \in \mathcal{C}_j$, $j \geq i$ of A (that is, G_B is a subgraph of G_A and $c_B(e) = c_A(e)$ for every $e \in E(B)$). Unfortunately, this step turned out to be time consuming and we could not do it for every case investigated; some manual adjustments needed to be done. In any case, at least the following operation was performed: a configuration $C \in \mathcal{C}_i$ was removed if $C \setminus e \in \bigcup_{j=i}^{k} \mathcal{C}_j$ for some $e \in E(C)$ or C is a super-configuration of some configuration from $\mathcal{C}_k \cup \mathcal{C}_{k-1}$.

Finally, let us discuss the role of "missing" edges, that is, edges for which $c(e) = 0$. The role of these edges is to point out places on the game board

that are available for Builder to play on. However, it is clear that if these places are already used by her (and Painter already coloured introduced edges), then it is better for Builder and she is in fact closer to her win than before. In other words, a configuration $C \in \mathcal{C}_i$ with some "missing" edges replaced by coloured ones must be in some later set $\mathcal{C}_j$, $j > i$. Hence, on the one hand, these "missing" edges can be removed which reduces the number of nonisomorphic configurations in $\mathcal{C}_i$. On the other hand, these "missing" edges make the merging process to be more restrictive and, as a result, fewer configurations are created. This trade off is tricky to handle so some parts have to be adjusted manually by the trial and error method. For example, to verify that $\overline{\mathcal{R}}(K_3, K_4) > 16$, we decided to keep "missing" edges until $\mathcal{C}_{11}$ is generated and from that point on we started neglecting them. An algorithm in this form took only 30 CPU minutes, that is, the running time decreased by a factor of more than $2 \cdot 10^6$ comparing to the original approach used in [13] for the same task.

Acknowledgment

This work used the facilities of the Shared Hierarchical Academic Research Computing Network SHARCNET. SHARCNET (`www.sharcnet.ca`) is a consortium of colleges, universities and research institutes operating a network of high-performance computer clusters across south western, central and northern Ontario. The infrastructure of SHARCNET consists of a group of 64-bit high performance Opteron and Xeon clusters with close to 20,000 CPUs.

References

1. J. Beck, *Achievement games and the probabilistic method*, Combinatorics, Paul Erdős is Eighty, vol. 1, Bolyai Soc. Math. Stud., 1993, pp. 51–78.
2. D. Conlon, *On-line Ramsey numbers*, SIAM J. Discrete Math. **23** (2009), 1954–1963.
3. J. Cyman and T. Dzido, *A note on on-line Ramsey numbers for quadrilaterals*, Opuscula Mathematica **34** (2014), 463–468.
4. J. Cyman, T. Dzido, J. Lapinskas, and A. Lo, *On-line Ramsey number of paths and cycles*, Electronic Journal of Combinatorics **22** (2015), #P1.15.
5. E. Friedgut, Y. Kohayakawa, V. Rödl, A. Ruciński, and P. Tetali, *Ramsey games against one-armed bandit*, Combin. Probab. Comput. **12** (2003), 515–545.
6. P. Gordinowicz and P. Prałat, programs written in C/C++ can be downloaded from `http://www.math.ryerson.ca/~pralat/` or `http://im0.p.lodz.pl/~pgordin/`.
7. J. Grytczuk, H. Kierstead, and P. Prałat, *On-line Ramsey numbers for paths and stars*, Discrete Mathematics and Theoretical Computer Science **10** (2008), 63–74.
8. A. Kurek and A. Ruciński, *Two variants of the size Ramsey number*, Discuss. Math. Graph Theory **25** (2005), no. 1–2, 141–149.
9. B. D. McKay, *`nauty` Users Guide (Version 2.5),* `http://cs.anu.edu.au/~bdm/nauty/`.
10. ———, *Practical graph isomorphism*, Congressus Numerantium **30** (1981), 45–87.
11. B. D. McKay and A. Piperno, *Practical graph isomorphism II*, J. Symbolic Computation **60** (2013), 94–112.
12. P. Prałat, *A note on small on-line Ramsey numbers for paths and their generalization*, Australasian Journal of Combinatorics **40** (2008), 27–36.

13. P. Prałat, $\overline{\mathcal{R}}(3,4) = 17$, Electronic Journal of Combinatorics **15** (2008), #R67, 13pp.
14. P. Prałat, *A note on off-diagonal small on-line ramsey numbers for paths*, Ars Combinatoria **107** (2012), 295–306.
15. S. Radziszowski, *Small Ramsey numbers*, Electron. J. Combin. Dynamic Survey DS1 (2014), 94, revision #14.
16. D. B. West, *Introduction to graph theory*, 2nd ed., Prentice Hall, 2001.

Institute of Mathematics, Lodz University of Technology, Łódź, Poland
E-mail address: pgordin@p.lodz.pl

Department of Mathematics, Ryerson University, Toronto, ON, Canada, M5B 2K3
E-mail address: pralat@ryerson.ca