20

Recent Developments in Bit-Parallel Algorithms

Pablo San Segundo, Diego Rodríguez-Losada and Claudio Rossi
Universidad Politécnica de Madrid
Spain

1. Introduction

A bit array (or bit vector, bitboard, bitmap etc. depending on its application) is a data structure which stores individual bits in a compact form and is effective at exploiting bit-level parallelism in hardware to perform operations quickly as well as reducing memory requirements. Working at bit level is nothing new: i.e. STL[1] for C++ has a *bitset* container as data type, and mapping pixels to bits or processes to priority queues in some operative systems are but two examples of an interminable list of applications where space requirements are critical.

However, to improve overall efficiency by bit-masking operations is *hard* in any scenario. One obvious reason for this is that bit vectors are compact data structures difficult to manipulate, all the more so since extracting information relative to a single bit of the array has an overhead which does not exist in a classical implementation. From a theoretical perspective there have been some important complexity results concerning bit-paralellism, where modern CPUs are seen as non deterministic Touring Machines with power limited to the size of its registers (denoted as w_{size}). In practice, bit-parallelism has become an important tool for domains such as string matching as in (Baeza-Yates R. and Gonnet G. H 1992), where the complexity of a linear algorithm is reduced by a factor w_{size}. It is important to note that these successes have not extended to more complex NP combinatorial problems in the general case, a key issue and a topic which has been a line of research of the authors in recent years.

A classical search domain for bit vectors has been board games, the origin of the term *bitboard*. In chess-playing programs, the bits in a 64-bit bitboard map to a particular Boolean property concerning the 64 squares of the chessboard (cf. Heinz E.A. 1997). One of the first systems to employ bitboards as the basic modelling unit is the KAISSA chess programming team in the Soviet Union during the late 1960s. Today almost all relevant chess programs employ this form of encoding and reason, at least partly, over a bit vector space.

This chapter covers the use of bit-parallelism as an AI tool to implement efficient search procedures. It focuses on fully bit encoded search domains, where declarative frame knowledge is mapped to bit vectors and procedural frame knowledge (i.e. basic transition operators etc.) is mapped to simple bitwise operations over a bit vector search space.

[1] STL: Standard Template Libraries

The material presented is structured in three parts. Section 2 covers exact (optimal) search. It focuses on a depth-first search algorithm to show the advantages and disadvantages of search in a bit vector space w.r.t. to a classical encoding, including experiments. Based on these experiments and recent work of the authors on the maximum clique problem (San Segundo P. et al. 2007) the section highlights the strength of simple graph models as a tool for implementing efficient bit parallel search procedures in general, and NP-hard problems in particular. At the end of the section the Boolean satisfiability problem and the N-Queens problem are suggested as new candidates for bit-parallel search.
Section 3 covers bit-parallel search in non-exact scenarios. In particular an efficient genetic algorithm for SAT is compared with an equivalent bit parallel version. The section also includes computational experiments. Section 4 describes two real life applications where bit-parallelism has been applied with success, taken from the vision and robotics domain. Conclusions as well as a brief discussion of future work are stated in Section 5.

2. Exact search in a bit vector space

This section covers exact (optimal) bit-parallel efficient search procedures. It is assumed that the search domain can be fully bit encoded and that a reasonable bit encoding has already been found. The subject of how to find one such bit representation for a particular domain is out of the scope of this Section (and of the Chapter itself). Rather, the Section focuses on implementation and complexity issues related to systematic bit-parallel search. As case study the maximum clique problem has been selected for a number of reasons that will be explained throughout the section.

2.1 Basic bit operators

A typical fully bit encoded search space maps bits to domain entities and states to a number of bit vectors which represent Boolean properties of these entities. Without loss of generality, it can be assumed that non Boolean properties which describe a particular state can be reduced to a collection of Boolean ones. In this scenario, a bit vector is a {0,1} collection of cardinality the number of domain entities. A possible declaration of this data structure in C language can be found in figure 1.

```
typedef unsigned long long  BITARRAY;

/*bit vector declaration*/
BITARRAY bitvector [Cardinality];
```

Figure 1. Declaration of a bit vector in C language

Since bit vectors map to sets, bitwise operations are needed to compute the fundamental operators related to set theory. Table 1 shows basic bitwise operations for sets using C style syntax (i.e., &, |, ^ and ~ map onto AND, OR, XOR and NOT respectively). Note that the last operator in Table 1 is not an assignment over sets A and B, but a truth assertion.
A fully encoded bit-parallel algorithm employs a bit vector (possibly more than one) to guide search in the bit space. In any systematic bit-parallel search procedure two classical

bitwise operators stand out over the rest: A) operator *LSB*[2] (alias Bit Scan Forward or simply Bit Scan) which finds the first 1-bit in a bit vector, and B) operator *PC* (*P*opulation *C*ount) which returns the number of 1-bits in a given bit vector. The former (or its counterpart MSB[3]) is typically used in node selection strategies whilst the latter is necessary for leaf node detection (typically the empty bitboard, $PC(BB)=0$). Notation throughout this paper includes an additional subindex to LSB or BB to make cardinality explicit (e.g. LSB_{64} refers to a bit scan over a 64 bit array).

$A \cap B \equiv A_{BB} \ \& \ B_{BB}$	$A-(A \cap B) \equiv (A_{BB} \wedge B_{BB}) \& A_{BB}$
$\overline{A} \equiv \sim A_{BB}$	$B-(A \cap B) \equiv (A_{BB} \wedge B_{BB}) \& B_{BB}$
$A \cup B \equiv A_{BB} \mid B_{BB}$	$A-B \equiv A_{BB} \& (\sim B_{BB})$
$(A \cup B)-(A \cap B) \equiv A_{BB} \wedge B_{BB}$	$A \supseteq B \Leftrightarrow B_{BB} \& (\sim A_{BB}) == \phi$

Table 1. Correspondence from set theory operators to bitwise operations written in C language.

Depending on the processor HW architecture and compiler used, both operators might be available as built-in functions or *intrinsics*, but their use is always restricted to the size of the CPU registers (w_{size}). The extension to bit vectors of cardinality higher than w_{size} is conceptually trivial but needs to be done carefully because the impact in overall efficiency is high. SW implementations of w_{size} LSB and PC are needed when they are not available as *intrinsics* and there are a large number of solutions available in literature (cf. Warren H.S. Jr 2002). For PC we recommend precomputation of a lookup table for all 16 bit possible combinations. For LSB a nice hashing solution for a 64 bit register CPU can be found in (1). MN is one magic number from a De Bruijn sequence such as 0x07EDD5E59A4E28C2. Computation BB&(-BB) isolates a single 1-bit and the *, >> operations constitute a perfect hash function for the isolani to a 6 bit index. For a more detailed explanation we refer the reader to (Leiserson, C. t al. 1998).

$$LSB_{64}(BB) = [(BB \,\&\, (-BB)) \cdot MN] >> 58 \tag{1}$$

A common assumption in bit encoded exact search models is that the benefits of parallelism at bit level have a counterpart in the overhead needed to extract information relative to a single bit in the compact bit array. This is, in fact, quite true in a general sense and is probably the reason why bit-parallelism has not attracted so much attention in AI real life applications as yet. This key issue is covered in the following subsection.

2.2 Complexity of bit scanning in bit-parallel systematic search

Finding a 1-bit in a compact bit array is an important overhead to be taken into account for efficient bit encoded exact search models. Worst case complexity for a naïve LSB_n computation is $O(n)$. A more efficient 16 bit direct lookup table implementation computes

[2] *LSB* stands for *L*east *S*ignificant *B*it
[3] *MSB* stands for *M*ost *S*ignificant *B*it

LSB_n in $O(\frac{4n}{w_{size}})$. For non bit-encoded models (e.g. an array indexed by the position of the element) the cost of a single LSB operation is in $O(n)$, clearly worse w.r.t. the bit model.
However, the situation changes when the problem is extended to finding the first k-bits in a bit set (alternatively the first k elements in a list). In this case, worst case complexity for lists is still in O(n) whereas, although it is possible to index the w_{size} blocks of bits, there is no getting over the $LSB_{w_{size}}$ complexity of finding a 1-bit in a particular block. Worst case computation, assuming $k <= w_{size}$ and a 16-bit direct lookup table implementation of $LSB_{w_{size}}$ is:

$$O(Bit\ Scan\ \ for\ k\ bits) = \frac{4(N-1)k}{W_{size}} \tag{2}$$

which grows linearly with the number of bits to find. Figure 2 illustrates this inherent complexity showing time results for finding the first 100 1-bits in a random generated population of size 5000 with varying densities.

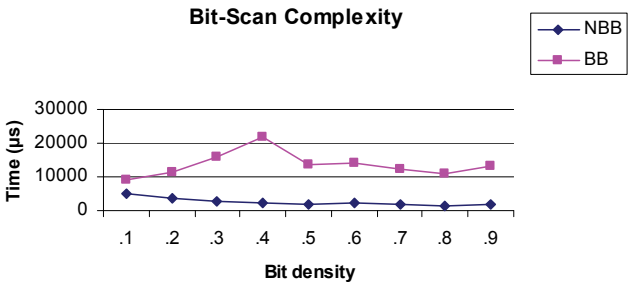


Figure 2. Different computing times for finding the first 100 elements in a randomly generated population of size 5000 after 1000 runs in a P-4 2.7GHz CPU. BB implements a compact bit array and NBB a list.

In the figure, BB stands for the compact bit array implementation, as opposed to a list or array made up of {0,1} integers. For the experiments the abovementioned 16 bit direct lookup table for LSB was employed. As expected, times for NBB remain reasonably linear with density whilst BB turns out to be more than 3 times slower in the general case. In (2), k=100, W_{size}=64, gives a 6 ratio difference in favour of NBB in the worst case, but average case for NBB is twice as fast since LSB will normally take two cycles and not four. As bit arrays become more and more sparse, average case for LSB decreases by another two factor since it takes one cycle to bit scan an empty block, so for d=0.1 the new ratio is $\frac{4 \cdot 100}{64 \cdot 4} \cong 1.5$.
Consider a bit vector space of states where a single bit vector BB_g guides some form of systematic search. This requires that every element of the set is expanded, so operator LSB_N must be called for every element of the set. Thus, the overall inherent complexity of the bit encoding is similar to finding the first k-bits in BB_g where k averages 1-bits for all states visited:

$$k = \frac{\sum PC(BB_g)}{Number\ of\ \ subproblems\ \ solved} \tag{3}$$

It is not clear that the benefits of computing transitions using bit-parallelism can outweigh this inherent bit scan complexity (e.g. in a brute force algorithm). In fact, the intuition is that additional bit encoded knowledge will be needed for efficient bit-parallel systematic search to improve a standard implementation. For some years now the authors have been interested in proving this statement for instances in the NP-complete class. Recent work in this line of research has led us to one such problem for the graph domain: the maximum clique problem. As a result, we have implemented a new complete bit-parallel general purpose algorithm which is one of the fastest general purpose algorithms at the moment (San Segundo P. et al, 2007). This result is important since it shows that bit-parallelism can be used as a tool to improve general purpose search algorithms for problems in NP. The following subsection focuses on this topic.

2.3 Bit encoded knowledge

The subsection is concerned with simple graphs. Simple graphs have a finite set of vertices V and a set E of pairs of vertices (x,y) called edges. Two vertices are said to be adjacent if they are connected by an edge. A subset of vertices such that every edge in W belongs to V is called a subgraph over G induced by W, and is written $G(E/W)$ or simply $G(W)$. A clique in G is an induced subgraph where every pair of vertices are bitwise adjacent. The *k-clique* problem consists in determining whether a clique of size *k* exists for a given graph and is well known to be NP-complete (Karp R.M. 1972). The corresponding optimization problem is the maximum clique problem (MCP), which looks for the largest possible clique in a given graph. MCP is NP-hard.

A typical efficient exact MCP algorithm uses a depth-first strategy to implement systematic search in a branch and bound scheme. Search takes place in a graph space starting with a small clique which gradually gets bigger and bigger as search advances. Recent examples of branch and bound algorithms for exact MCP are (Pardalos P.M. and Xue J. 1994) and more recent (Tomita E. and Kameda, T. 2006) amongst others. Figure 3 shows a primitive branch and bound MCP algorithm. It receives as input a simple graph *G* and returns the size of the largest possible clique in variable max_size. $N_G(v_i)$ is the neighbour set of vertex v_i in G and contains all vertices in G adjacent to v_i.

```
Initialization: U=Input Graph G, size=0

function clique(U, size)
Step 1:Leaf node (|U|=0)
        1. if (max_size<size) max_size = size
        2. return
Step 2:Fail upper bound (size +|U|<max_size)
        1. return
Step 3 For every node in U
        1. Select vertex (v_i)
        2. U:= U - {v_i}
        3. clique(U ∩ N_G(v_i) , size+1)
```

Figure 3. A primitive exact branch and bound algorithm for MCP.

Simple graphs have {0,1} adjacency matrices where element A_{ij} is 1 if there is a corresponding edge between vertex *i* and vertex *j* in the graph and 0 otherwise. As a consequence, binary matrices map nicely to bit arrays of size the number of nodes of the graph (e.g. one bit array per row).

For a full encoding of the MCP search space, an additional bit array BB_{guide} is needed to guide the search, mapping the set of vertices of the graph at the current node. Initially BB_{guide} starts with all bits to 1 corresponding with the initial input graph. An empty BB_{guide} is a leaf node whilst vertices expanded in any path from root to leaf form a clique in *G* (see figure 4).

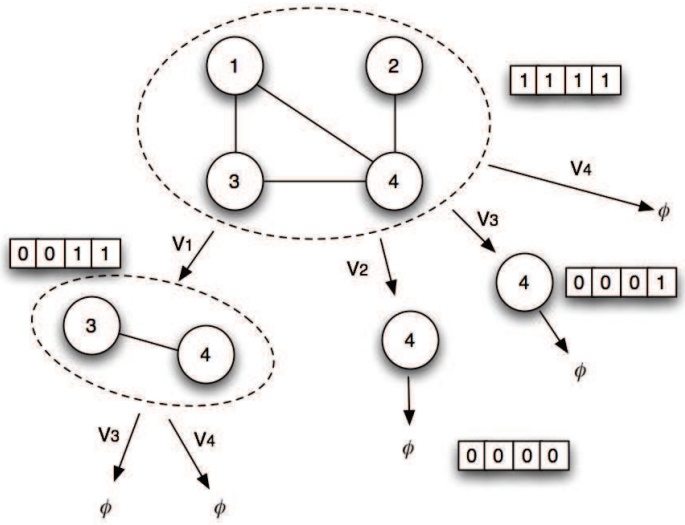


Figure 4. An example of MCP search in a bit encoded graph space. A single bit array guides the search. The bit encryption maps the i-th vertex of a graph to the i-th bit in the bit array. Every path from root to leaf node is a clique.

At every node bit scanning is needed during vertex selection for expansion, an overhead which has an important overall impact w.r.t. a non bit_parallel implementation. To validate this statement a number of tests have been carried out with a naïve brute force MCP algorithm denoted BBN-MCP (labelled BB in figure 5), and an equivalent non bit-parallel implementation N-MCP (labelled as *list* in figure 5). Both implementations use depth first systematic search to explore the full space without any pruning strategy and vertices are selected lexicographically.

Figure 5 shows time results for a number of randomly generated graph instances of different sizes and densities. In this systematic lexicographic brute force scenario, results indicate that the complexity of bit scanning at every node far outweighs the advantage of computing graph transitions efficiently using bitwise operations.

Things change when knowledge gathered during early exploration in depth-first search is bit encoded to prune the space later on. In MCP, strong efficient upper bounds on the size of the maximum clique for any graph can be computed through coloring of the graph vertices.

Classical vertex coloring of a graph $G=(V,E)$ is just a way to partition set V into disjoint subsets C_i of same color vertices. The restriction behind coloring is that only non adjacent vertices can be *painted* with the same color. Let C_i be the *i-th* color set of a possible *k-coloring* for G (see 4).

$$\bigcup_{i=1}^{k} C_i = V, \quad \bigcap_{i=1}^{k} C_i = \phi, \quad \omega(V) \leq k \tag{4}$$

where $\omega(V)$ is common notation for the size of the largest possible clique in *G*. The best upper bound by vertex coloring for $\omega(V)$ is the graph chromatic number i.e., the minimum number of colors needed to paint the graph.

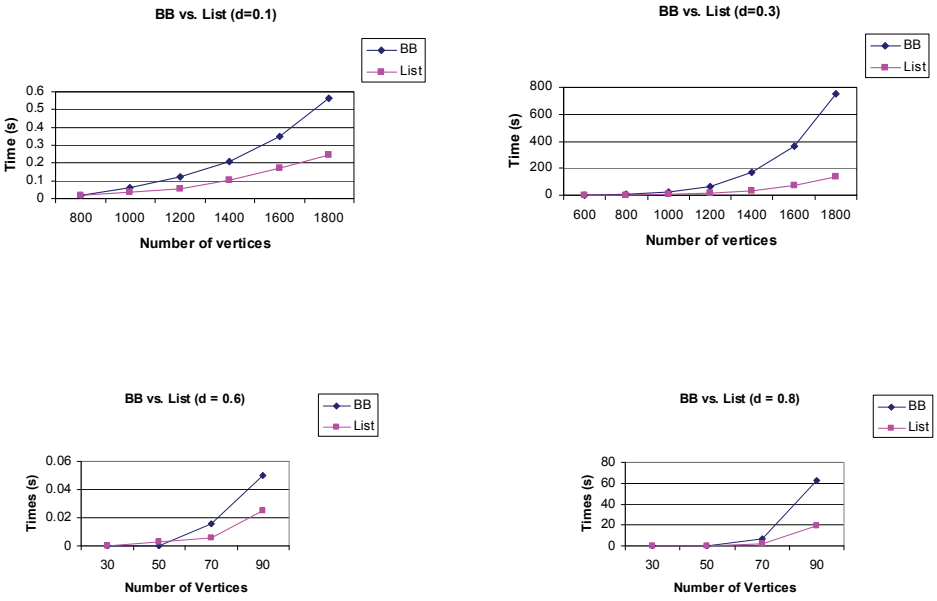


Figure 5. Time results for a bit-parallel (BB) and a classical (List) naïve brute force MCP algorithm.

Since optimal coloring is also in NP, efficient MCP complete algorithms use some form of greedy coloring strategy to prune the search space. There are many possible such strategies and an adequate survey is out of the scope of this article.

Of interest in this paper is the fact that previous naïve BBN-MCP implementation turns out clearly superior to N-MCP when a typical coloring scheme is added. The coloring implemented is a standard technique commonly used which is in $O(n^2)$, and runs w_{size} times faster in BBN-MCP than the non bit-parallel implementation. The impact of the pruning strategy for MCP is so big in the majority of cases that its computation becomes critical for overall efficiency. Figure 6 shows times for N-MCP and BBN-MCP when the coloring scheme is included. The situation is now reversed; bit scanning overhead is clearly surpassed by the benefits of bitwise coloring.

Figure 6. Time results for a bit-parallel (BB) and a classical (list) implementation of a naïve MCP algorithm with a classical vertex coloring strategy to establish bounds.

2.4 Graph models for bit-parallel search

The interest of this article is focused in efficient bit-parallel NP algorithms. In the authors' view, two very promising lines of research can be undertaken. In the first place, results presented in the previous subsection make MCP a promising tool for implementing bit-parallelism in other NP problems. A survey on our very efficient bit-parallel MCP algorithm can be found in our recent work (San Segundo P. et al. 2007). As has been said, *k-clique*, the corresponding non optimization version of MCP, is an NP-complete problem so it is certainly conceivable that problems with a reasonably benevolent reduction to *k-clique* can be efficiently solved using some form of bit-parallelism.

A second and more general line of research can be found in the intrinsic binary nature of simple graphs, which make them a very important tool by themselves to exploit bit-parallelism in search. The reason behind this is that the binary adjacency matrix of such graphs allows for a simple and clear mapping of relations to bits. Moreover it also facilitates the bit encoding of additional domain dependent knowledge, which can then be computed by efficient bitwise operations.

Following this second line of research, our attention has recently shifted to bit-parallelism in the Boolean satisfiability problem (commonly known as SAT). At the moment we have implemented a number of graph models to represent clause information with, as yet, modest but highly encouraging results. We note that today's fastest general purpose SAT algorithms do not employ reduction to a graph space; it is actually a more common practice to reduce other problems to SAT (e.g. (Kautz, H. and Selman, B. 1998) is a very efficient planner which solves a graph plan in a SAT space). Some NP-hard problems taken from board games have also an interesting reduction to simple graphs which might need reviewing from a bit-parallel perspective. One such example is N-Queens which aims to place N queens in an empty NxN square board such that they do not attack each other. More complicated scenarios include an initial non empty board (e.g. with a pawn on a particular square). A possible graph model for such scenarios maps vertices to squares in the board and places an edge between two squares if a queen placed on any one of them attacks the other.

Besides optimal search procedures we have also done some recent research on bit-parallelism in evolutionary algorithms. In this case the aforementioned intrinsic complexity of bit scanning is not necessarily a key issue because candidate solutions can be generated using other means (e.g. a permutation index). The issue of bit-parallelism in suboptimal search procedures is the focus of the next section.

3. Evolutionary algorithms

The term *Evolutionary Computation* denotes a class of population-based heuristic search techniques inspired by Darwin's principle of evolution in nature. Starting from a set of candidate solutions, called *population*, an evolutionary algorithm (EA) generates a new population of candidate solutions by means of operations called *selection*, *recombination*[4] and *mutation*, applied to the existing population. This step is called a *generation*. Generation after generation the population of candidate solutions *evolves* toward good solutions of the problems at hand. In analogy with natural environments, candidate solutions are also called *individuals*. Each individual is represented by a *chromosome*, which is an encoding of a candidate solution. Every individual has associated a *fitness* that is a measure of its quality, i.e., how good the individual is in solving the given problem. The term Evolutionary Computation denotes a whole family of techniques, which differ on some aspects from the evolutionary loop. Evolutionary Strategies, Genetic Algorithms, Genetic Programming, and many other evolutionary based search techniques apply the same basic concepts, but differ on how the selection, recombination, mutation, encoding of individuals and survivor selection operations are implemented. A detailed description of all the aspects of the evolutionary computation galaxy goes beyond the purpose of this work. For a good survey we refer the reader to (Eiben A.E. and Smith J.E., 2002).

Because of the natural representation of candidate solutions as string of bits, where each bit represents the truth value of the corresponding variable, SAT is the typical problem that can be approached using standard genetic algorithms (GAs), i.e. evolutionary algorithms based on bit-string representation of chromosomes. However, it was observed that since EAs do not use domain dependent knowledge, they may not outperform well tuned problem specific algorithms. This observation has been experimentally confirmed, and justifies the fact that all evolutionary algorithms for SAT proposed in the recent years have incorporated heuristic information. EAs for SAT can be roughly divided into three main classes depending on the way they use knowledge: EAs that encode knowledge into the fitness function, in the genetic operators and those that use the MAX-SAT fitness function (see below) and add local search to improve the quality of individuals. Usually EAs for SAT adopt the bit representation, since this is the most natural representation for this problem. However, EAs based on other representations have been used, like clausal representation, floating point, and path representation.

Several different evolutionary algorithms have been proposed for the SAT problem, varying in the representation and/or fitness function. For an exhaustive survey we refer to (Gottlieb et al., 2002). In the following, we will analyze the ASAP algorithm (*Adaptive evolutionary algorithm for the Satisfiability Problem*), which is one of the best evolutionary algorithms for the Sat problem, and proved to be competitive with the best non-evolutionary algorithms (Rossi, C. Et al., 2000).

The ASAP algorithm is a *(1+1)-evolutionary strategy* enhanced with a local search step and a memory of past states that is used to escape from local minima and to dynamically adapt the mutation parameter. In a *(μ+λ)-strategy* the population consists of μ individuals. At each generation, λ new individuals are generated, and the new population is formed by the best μ among the $(\mu + \lambda)$ individuals. In ASAP, since the population is formed by only one candidate solution, there is no recombination operator, and only mutation is used to

[4] Recombination is also known as *crossover*.

generate the offspring (see Fig. 7, left). Mutation consists in changing the value of a bit of the chromosomes chosen at random with a certain probability, called *mutation rate*.

3.1 Description of the ASAP algorithm

In ASAP, to each new individual a local search procedure called *Flip Heuristic* is applied. The technique of using local search operators in combination with evolutionary algorithms is called *Evolutionary Local Search* or *Memetic* search.

```
PROCEDURE ASAP
  randomly generate chromosome C
  apply Flip Heuristic to C
  WHILE (not termination condition) DO
    BEGIN
      C0=C      /* store parent C */
      apply adaptive mutation to C
      apply adaptive Flip Heuristic to C
      compute fitness of C
      ID(fitness C < fitness C0)
        C=C0   /* discard new C */
      ELSE
        UPDATE_TABLE(C)
    END
END PROCEDURE
```

```
PROCEDURE UPDATE_TABLE
  BEGIN
    IF (fitness C > fitness CO)
      BEGIN
        empty table T
        add C to table T
        unfreeze all genes
      END
      ELSE /* fitness CO=fitness C*/
        BEGIN
          add C to table T
          IF (table T full)
            BEGIN
              compute frozen genes
              adapt mutation rate
              empty table T
            END
        END
END PROCEDURE
```

Figure 7. ASAP pseudo-code

Roughly, it consists in the application of genetic operators to a population of local optima produced by a local search procedure. The Flip Heuristic consists in repeatedly flipping one bit in a randomly generated sequence, and keeping the change if this leads to an increment of the fitness function (i.e., more clauses becomes satisfied then becomes unsatisfied). When no increment has been obtained, the procedure terminates.

The choice of an appropriate fitness function is very important in the design of an evolutionary algorithm. ASAP adopts the most used fitness function for the Sat problem in EAs, called MAXSAT. The MAXSAT formulation assumes that the Sat problem is expressed in conjunctive normal form, i.e. it is a conjunction of *m clauses* c_i, *i=1..m*, each of which is a disjunction of *literals* (a variable or its negation).

$$f(x) = c_1(x) \wedge \ldots \wedge c_m(x), \quad c_i = (l_{i1} \vee \ldots \vee l_{ik})$$

where *x* is the array of the Boolean variables. In the MAXSAT formulation, the fitness value is equivalent to the number of satisfied clauses, i.e.,

$$f_{MAXSAT}(x) = val(c_1(x)) + \ldots + val(c_m(x)),$$

where *val* $(c_i(x))$ maps the truth value of the i-th clause into an integer value 1 when the clause is true and 0 when it is false. In this way, the range of the function changes from {*true,false*} to {*0..m*}. Note that in this formulation the optimum value is known in advance, since the formula is satisfied when all its *m* clauses evaluates to 1.

ASAP is provided with a memory of past states. This is used to escape from local minima in a twofold way. Observe that at each generation the algorithm produces a local optimum.

Suppose the local search procedure directs the search towards similar (that is, having small Hamming distance) local optima having equal fitness function values. Then we can try to escape from these local optima by prohibiting the flipping of some genes and by adapting the probability of mutation of the genes that are allowed to be modified. To this aim, ASAP uses the following technique inspired on TABU search (see Fig. 7, right). At each step, a table T of size *k* is filled with chromosomes having best fitness. If the best fitness increases then the table is emptied. When the table is full, the chromosomes are compared gene-wise. Those genes that do not have the same value in all the chromosomes are labelled as *"frozen"*. The information contained in T is used for adapting the search strategy during the execution as follows. Each time T is full, the mutation rate is recomputed setting it to the value mutation_rate = ½ · (nº. of frozen variables)/n (thus, 0 < mutation_rate < 0.5), and the flipping of frozen genes is prohibited. The rationale behind these two actions is the following. If table T becomes full it means that the search strategy has found for k times best chromosomes with equal fitness. A non-frozen gene has the same value in all these chromosomes. This indicates that the search directs often to local optima containing the same values of such genes. Therefore in the next iteration we allow to flip only not frozen genes in order to reach search points far enough from the attraction basin of those local optima. The mutation rate is chosen in such a way that the lower the number of not frozen genes is, the higher the probability will be to flip them, since a strong basin of attraction, requires a higher probability of generating individuals that are very different ("far") from its parent. The term 1/2 is used to keep the mutation rate smaller than or equal to 0.5.

Although the most obvious way to represent a solution candidate for SAT is a bit string of length n, where every variable is associated to one bit, in the original implementations of ASAP this was for simplicity encoded as an array of integer values, taking the value 0 or 1. A new version of ASAP has been implemented adopting the bit board representation, in order to exploit the benefits of bit-parallelism. We will refer to the new version as ASAP-BB. In order to analyze the benefits of adopting the new representation let us analyze in detail the computational cost of producing a new solution ASAP.

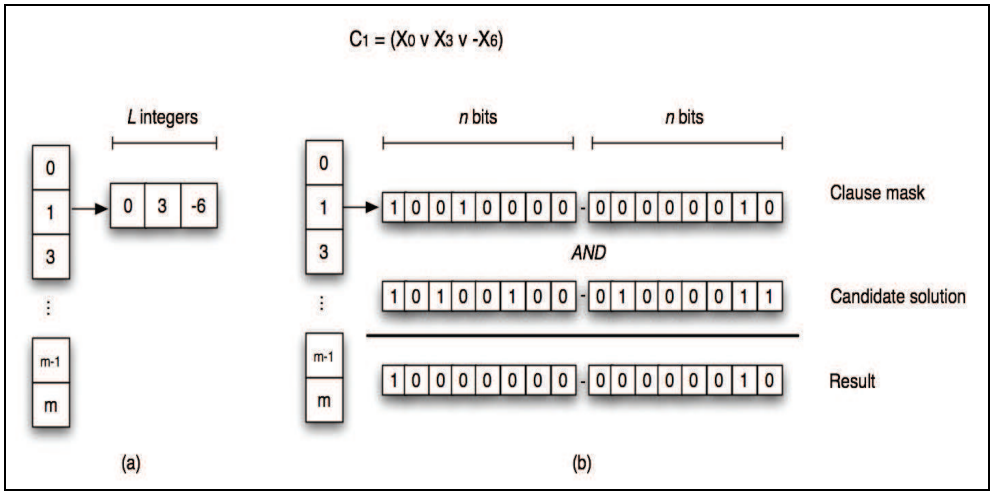


Figure 8. Clause representation: (a) array of indexes; (b) bit arrays.

Let m be the number of clauses, n the number of variables and L the average clause length (number of literals) of a given SAT instance. As mentioned before, each time an offspring candidate solution is generated, it is repeatedly improved by a series of bit flips. Each time a bit is flipped the fitness function must be recomputed in order to check whether the change leads to an improvement. This is the most expensive operation and is repeated several times. Computing the fitness function implies looping through the clauses and re-computing them by assigning to its literals the value of the corresponding variable of the solution. In ASAP, a solution is represented as an array of n integers, and a clause is represented as an array of integer values, containing the indexes of the variables contained in the clause (see Fig. 8 (a)). Thus, on average, each clause computation involves L integer operations. The cost of a fitness function evaluation is $m \cdot L$.
In ASAP-BB, a solution is encoded as a bit vector of length $2n$, containing the values of the variables in the first half, and their negation in the second half. A clause is encoded as a bit vector of $2n$ bits, called clause mask. The first n bits have their value set to *'1'* if its position corresponds to the index of a non-negated variable of the clause, and *'0'* otherwise. The second n bits are set in the opposite way: bits are set to *'1'* in correspondence of negated variables (see Fig. 8 (b)). Thus, the evaluation of a clause is performed as a bit-wise *AND* operation between a solution and the clause mask which allows exploiting bit parallelism. The cost W of such operation depends on the word length and the number of variables of the problem at hand:

$$W = N\_WORDS = \left\lceil \frac{2n}{WORD\_LENGTH} \right\rceil \tag{5}$$

and the total cost of evaluating the fitness function will be $m \cdot W$.
Thus, considering that the fitness calculations are the core of the algorithm, and everything else is kept unchanged, the expected speedup of ASAP-BB w.r.t ASAP is

$$Speedup = \frac{mLF}{mWF} = L/W \tag{6}$$

where F is the total number of fitness evaluations. The expected speedup depends on the average clause length and on the number of variables, the latter determining the size of the bit array.
As far as space is concerned, a similar analysis can be performed. The total storage space for a clause in ASAP is $L \cdot m$ integers while in ASAP-BB it is $2 \cdot n \cdot m$ bits. Assuming an integer has a size of four bytes, the space occupation ratio is

$$Space = \frac{2mn/8}{4mL} = \frac{n}{16L} \tag{7}$$

3.2 Experiments

In order to validate the previous analysis of time and space complexity, we have performed a series of tests on a set of standard benchmark instances, all satisfiable. Instance family

3SAT was the first used to test different EA-based algorithms for SAT (cf. Bäck T. et al., 1998). These instances are random 3-SAT benchmark instances with $m/n = 4.3$[5] generated using the *mkcnf*[6] generator using the *forced* option to ensure that they are satisfiable. Instance families *II, Aim, Jnh, Par* are taken from the 2nd DIMACS challenge on cliques, coloring and satisfiability (Johnson D. and M. Trick, 1996). The *Aim* family contains artificially generated 3-SAT instances and are constructed to have exactly one solution. Family *Par* instances arise from a problem in learning the parity function. The *Jnh* instances are randomly generated and have a varying clause length. Instances *II* arise from the "Boolean function synthesis" problem and are used in inductive inference.
Table 2 reports the results of the tests performed. In order to compute the real speedup, times for ASAP and ASP-BB are averaged after 10 runs on every instance[7]. The speedup values have been computed averaging all the results of instances with similar properties (i.e. m and L values).
The table shows that the measured speedup is in accordance with the analysis performed, with small differences that are, in general, smaller than the standard deviation σ, and thus are not statistically significant. Note that the Par and II instances have a clauses/variables ratio that is disadvantageous for the bit array representation.
As far as the space ratio is concerned, the bit vector representation saves space w.r.t. the plain integer array representation only when the number of literals remains low. Worst case space ratio occurs for II instances, with a 70% increment approx.

4.The geometric correspondence problem

In this final section we present a survey on recent work done by the authors where bit-parallelism has been applied to a real life problem with success. More specifically, an exact bit parallel algorithm for the maximum clique problem has been conveniently applied to solve the correspondence problem between two sets of geometric entities, also known as relational structure search (Bomze et al., 1999) in the vision domain or the data association problem (Siegwart & Nourkbash, 2004) in mobile robotics. The section starts with a description of an adequate representation of the problem for reduction to MCP and ends with some experiments with real data.

4.1 Description

Given two sets of geometric features (i.e. points, segments etc.) the aim is to find the best correspondence between both sets. If a weighted graph of geometric relationships is built in each set, with a relationship (e.g. a metric) established between every two features, the problem becomes that of finding the Maximum Common Subgraph (MCS) between them. MCS is known to be NP-hard, and its solution becomes even more difficult in the case of noisy scenarios, when simplifying hypothesis or approximations cannot be applied. This

[5] The 4.3 clause/literal ratio is such that instances generated with lower ratio (the *underconstrained* region) almost always have solutions. Those generated with higher ratio (the *overconstrained* region), almost always have no solutions. Recent works have identifed that hard random k-SAT instances lie in such backbone, also know as *phase transition* region.
[6] See ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances
[7] ASAP is non-determinist, hence multiple runs must be performed in order to obtain an average behaviour.

setting occurs very often in many applications, as comparing fingerprints, mobile robot global localization, computer vision, pattern matching, etc.

Family	*Instance*	*No. of variables (n)*	*No. of clauses (m)*	*Average clause length (L)*	*BB length (W)*	*Expected speedup*	*Average speedup*	σ	*Space ratio*
3SAT	1	30	129	3.00	1	3.00	2.959	0.092	0.63
	2	30	129	3.00	1	3.00			
	3	30	129	3.00	1	3.00			
	4	40	172	3.00	2	1.50	1.551	0.178	0.83
	5	40	172	3.00	2	1.50			
	6	40	172	3.00	2	1.50			
	7	50	215	3.00	2	1.50	1.446	0.145	1.04
	8	50	215	3.00	2	1.50			
	9	50	215	3.00	2	1.50			
Aim	50-3_4-1	50	170	3.00	2	1.50	1.497	0.022	1.04
	50-3_4-2	50	170	3.00	2	1.50			
	50-3_4-3	50	170	3.00	2	1.50			
	50-3_4-4	50	170	2.99	2	1.49			
	50-6_0-1	50	300	3.00	2	1.50			
	50-6_0-2	50	300	2.99	2	1.50			
	50-6_0-3	50	300	2.99	2	1.50			
	50-6_0-4	50	300	3.00	2	1.50			
II	8a1	66	186	2.42	3	0.81	0.870	-	1.70
Jnh	1	100	850	5.17	4	1.29	1.311	0.014	1.21
	201	100	800	5.19	4	1.30			
	12	100	850	4.91	4	1.23	1.438	0.015	1.27
	204	100	800	4.89	4	1.22			
	205	100	800	4.89	4	1.22			
	210	100	800	4.89	4	1.22			
	213	100	800	4.88	4	1.22			
	218	100	800	4.88	4	1.22			
	7	100	850	4.89	4	1.22			
Par	8-1-c	64	254	2.88	2	1.44	1.410	-	1.39
	8-2-c	68	270	2.89	3	0.96	0.930	0.033	1.47
	8-3-c	75	298	2.90	3	0.97			1.62
	8-4-c	67	266	2.89	3	0.96			1.45
	8-5-c	75	298	2.90	3	0.97			1.62

Table 2. Results for SAT tests using ASAP and ASAP-BB. Times are averaged after 10 runs.

The relationships between geometric features, also called constraints, are pose invariant relationships that relate both features. If the sets are composed by 2D features, the constraints could be:

- **Distance**: Euclidean distance between two points.
- **Angle**: Angle between two non parallel segments.
- **Distance**: Shortest (perpendicular) distance from point to segment.
- **Distance**: (Shortest) distance between two parallel segments.

A weighted graph reduction to MC capturing such relationships in each set would map vertices to the elements of the set and weighted edges to the constraints.

This geometric correspondence problem allows a more convenient formulation (Bailey 2002) under the MCP paradigm. Figure X illustrates this idea employing two sets L and O. The former contains N features called landmarks (L_1,…, L_n), and the latter is the observation set containing M features called observations (O_1,…, O_m). Instead of searching the MCS between those two sets, a new graph called the association graph is defined, in which the nodes represent each possible landmark-observation pairing. Thus, the number of vertices of the correspondence graph is a priori NxM.

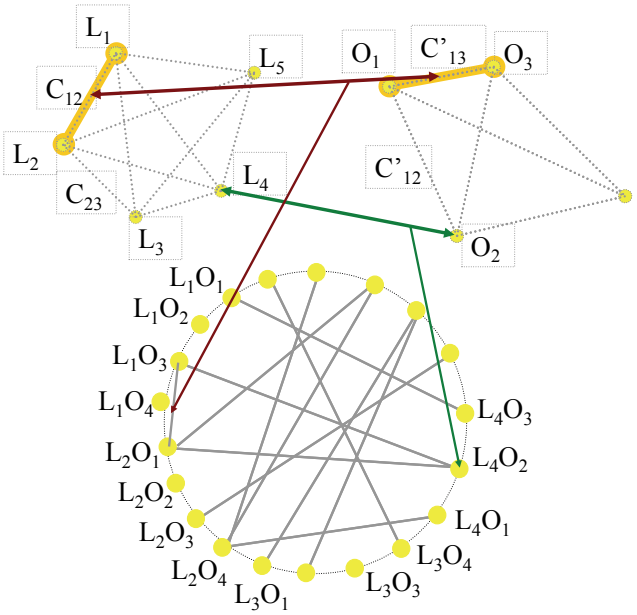


Figure 9. The geometric correspondence problem as an MCP search in an associations graph

The edges of the association graph are defined by checking pairs of constraints between the landmark and the observation initial graphs. If a constraint that relates two landmarks in the landmark set (e.g. constraint C_{12} relates landmarks L_1 and L_2) is compatible with a constraint that connects two observations in the observation set (in the example C_{13}' that connects O_1 and O_3), then L_1O_3 and L_2O_1 vertices in the association graph are connected. In this new association graph, the correspondence problem is reduced to finding the maximum clique, equivalent to maximizing joint compatibility. Once the problem has been reduced to MCP we have applied bit-parallelism in a similar way as described in section 2.

4.2 Experiments

Two different sets of experiments with the proposed solution to the geometric correspondence problem have been carried out: image matching and mobile robot global localization. In both scenarios, the required processing time has been the main output. Solutions obtained in all cases are optimal. The results have been compared with a finely tuned version of the MCS algorithm, running in the same computer.

4.2.1 Image matching

In this experiment, a large aerial image of our city, Madrid has been selected because its repetitive structure. The total image size is 1806x1323 pixels, and each pixel represents approximately 0,4m, so the area covered is about 792x580m. Figure X shows one ninth of such image and it can clearly be observed that its "texture" is quite repetitive making the recognition of a partial image hard for the human eye. Given a partial subimage with unknown position and orientation, the problem studied is to find the correspondence in the full image. Pre-processing includes corner detection as in (Rosten & Drummond, 2005, Rosten & Drummond, 2006) applied to both images to extract relevant points that can be used as features. It has to be noted that due to different lighting conditions, noise, dynamic objects, not necessarily the same corners are detected in both images (see figure X). In such a noisy scenario, the exact solution could be the only way to guarantee robustness.

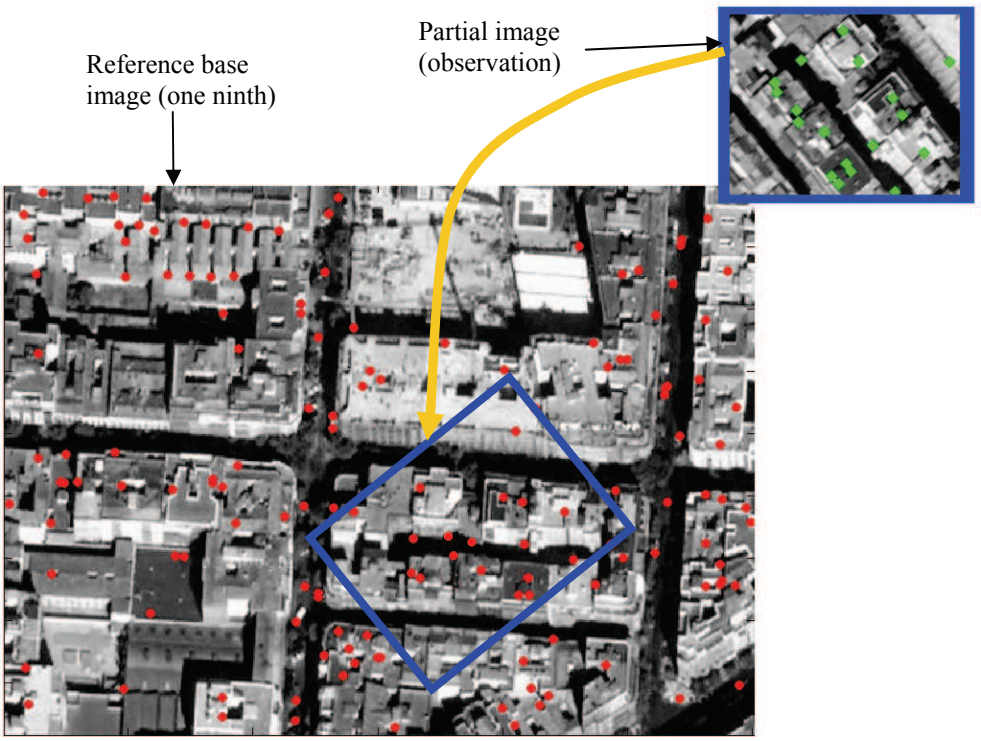


Figure 10. Computer vision pattern matching

The table shows that processing time quickly increases with the number of landmarks and observations for the MCS algorithm, but BE-MCP remains reasonably low. Furthermore, the variance of BE-MCP is quite small w.r.t. MCS, which increases rapidly with the size of the problem.
Table 3 shows processing times for both algorithms in different settings of landmark-observation pairs, depending on the corner detection threshold chosen. Each setting is repeated 10 times with different random observation subimages, and the average, best and worse times are shown in the table. The algorithms were implemented in C++ and tests were run on a P4 2,6GHz laptop.

Time (s)	Algorithm	MCS				BE-MCP			
	Number of landmarks	1023		1464		1023		1464	
Number of observations	7	3,22	8,83 0,55	16,24	31,75 4,67	0,09	0,09 0,06	0,18	0,20 0,17
	10	10,89	23,28 4,17	41,44	87,45 14,66	0,16	0,17 0,14	0,35	0,36 0,33
	15	32,83	59,70 6,81	86,04	207,25 28,64	0,37	0,39 0,36	0,79	0,95 0,74

Table 3. Comparison of processing times for different settings between a bit-parallel MCP search algorithm and an MCS solver

4.2.2 Mobile robot global localization

Finding a robot position in a given map with only observations on local features is called mobile robot global localization. If the map contains a set of geometric entities such as segments (e.g. the environment walls), the observations of the robot will also be modelled as such, but due to noise, dynamic objects (ie.g. people) and sensor limitations, these observations can be also noisy and incomplete.
Using the MCP approach we have solved the global localization problem using real data from our interactive tour-guide robot called Urbano (see figure 11). The reference maps were built in real time with an EKF based SLAM algorithm (Rodriguez-Losada et al. 2006). This time comparisons between BE-MCP and MCS were carried out under different levels of noise, in a map composed by approximately 350 features, with observation sets made up of between 25 and 30 observations. In the observation sets, a variable number of spurious observations were allowed ranging from 5 to 14 (the latter being almost 50% of the total).
In this case the average time required for MCS increased with the number of noisy observations, but the time required for BE-MCP remained constant. Furthermore, the variance in the BE-MCP also remained constant, while the variance in MCS increased degrading the worst case performance.

5. Conclusions and future work

Using bit-parallelism to implement efficient search algorithms raises a number of fundamental questions which the authors have tried to cover to some extent in this chapter. In the first place it has been shown that scanning for 1-bits in a compact bit arrays is an intrinsic overhead which must be taken seriously into account, especially in systematic

search procedures. In the second place emphasis has been laid on the importance of simple graph models because of their inherent binary adjacency matrix-bit array mapping. This fact ensures a natural bit encoding of frame knowledge as well as facilitates bit encoding of additional domain dependent knowledge. Following this line of research, recent work done by the authors on the maximum clique problem has revealed that its particular nature makes it a very good tool to implement efficient bit parallel algorithms for problems in NP.

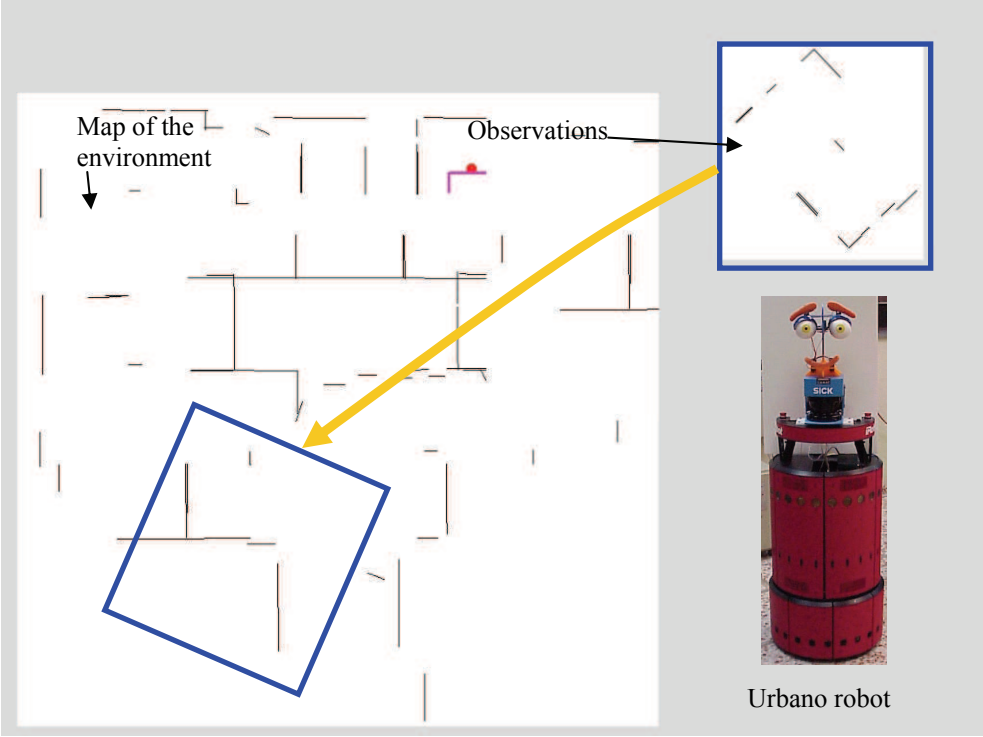


Figure 11. Mobile robot global localization

Attention has also been paid to bit-parallelism in suboptimal search. The analysis and experiments with the ASAP genetic algorithm have shown that bit parallelism can be beneficial for the SAT problem depending on the problem instance and of the specific data structures used to manage the bits. An optimized bit array structure would allow achieving even better performances than the ones obtained in the experiments performed here.
Finally the authors present a brief survey on two real life applications where bit-parallelism has proved successful.

6. Acknowledgements

This work is funded by the Spanish Ministry of Science and Technology (Robonauta: DPI2007-66846-C02-01) and supervised by CACSA whose kindness we gratefully acknowledge.

7. References

Rosten E. and Drummond T., 2005. Fusing points and lines for high performance tracking. IEEE International Conference on Computer Vision. Oct 2005. Vol 2. pp 1508-1511.

Rosten E. and Drummond T., 2006. Machine learning for high-speed corner detection. European Conference on Computer Vision.

Rodriguez-Losada D., Matia F., Galan R. Building geometric feature based maps for indoor service robots. Elsevier: Robotics and Autonomous Systems. Volume 54, Issue 7 , 31 July 2006, Pages 546-558

Bomze I.M., Budinich M., Pardalos P.M., Pelillo M., 1999. HandBook of Combinatorial Optimization, Supplement Vol A. Kluwer Academic Publishers, Dordrecht, 1999 pp.1-74.

Bailey T. 2002, Mobile Robot Localisation and Mapping in Extensive Outdoor Environments. PhD thesis. Australian Centre for Field Robotics, University of Sydney.

Siegwart R., Nourkbash. I. An Introduction to Autonomous Mobile Robots, MIT press, 2004.

Eiben A.E. and Smith J.E., 2002. Introduction to Evolutionary Computing, Springer, 2002.

Gottlieb J., Marchiori E. and C. Rossi 2002, Evolutionary algorithms for the satisfiability problem. Evolutionary Computation Vol. 10, Nr. 1, pp. 35-50, 2002.

Rossi C., Marchiori E. and Kok J.N. 2000, An adaptive evolutionary algorithm for the satisfiability problem. In Proceedings of ACM Symposiumn Applied Computing, pages 463–469, 2000.

Bäck T., Eiben A.E. and M. Vink 1998 A superior evolutionary algorithm for 3-SAT. In Saravanan, N., Waagen, D., and Eiben, A., editors, Proceedings of the Seventh Annual Conference on Evolutionary Programming. Lecture Notes in Computer Science, Volume1477, pages125–136, Springer, Berlin, Germany, 1998.

Johnson D. and M. Trick editors 1996, Cliques, Coloring and Satisfiability. AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26, 1996.

Baeza-Yates R. and Gonnet G. H 1992, A new approach to text searching. Commun. ACM, 35(10), pp:74–82, (1992).

Heinz E.A. 1997, How DarkThought plays chess, ICCA Journal, 20(3): pages 166-176, 1997.

San Segundo P., Rodriguez-Losada D., Galán R., Matía F. and Jiménez A. 2007, Exploiting CPU bit parallel operations to improve efficiency in search. International Conference on Tools for Atificial Intelligence (ICTAI 07). Patrás, Grecia, Octubre 29-31, 2007.

Leiserson, C., Prokop, H., and Randall, K. (1998). Using de Bruijn sequences to index a 1 in a computer word. See: http://supertech.csail.mit.edu/papers/debruijn.pdf.

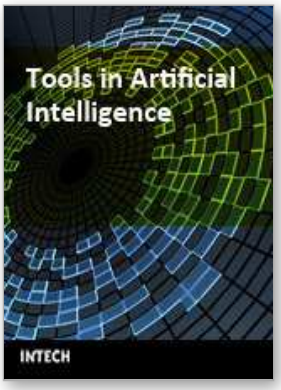
Karp R.M. 1972. Reducibility among Combinatorial Problems. Editors: R.E. Miller, J. W. Thatcher, New York, Plenum, pp: 85-103 (1972).

Pardalos P.M. and Xue J. 1994, The maximum clique problem. Global Optimization. 4: pp. 301-328, (1994).

Tomita E. and Kameda, T. 2006. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. Journal of Global Optimization (37), Springer, pp: 37:95-111 (2006)

Kautz, H. and Selman, B. 1998. BlackBox: A new approach to the application of theorem proving to problem solving. En AIPS98 Workshop on Planning as Combinatorial Search, pag. 58-60, Junio 1998.

Warren H.S. Jr 2002, Hacker´s Delight. Addison-Welsey 2002.



Tools in Artificial Intelligence

Edited by Paula Fritzsche

ISBN 978-953-7619-03-9
Hard cover, 488 pages
Publisher InTech
Published online 01, August, 2008
Published in print edition August, 2008

This book offers in 27 chapters a collection of all the technical aspects of specifying, developing, and evaluating the theoretical underpinnings and applied mechanisms of AI tools. Topics covered include neural networks, fuzzy controls, decision trees, rule-based systems, data mining, genetic algorithm and agent systems, among many others. The goal of this book is to show some potential applications and give a partial picture of the current state-of-the-art of AI. Also, it is useful to inspire some future research ideas by identifying potential research directions. It is dedicated to students, researchers and practitioners in this area or in related fields.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821