Did this paper address an important issue? Explain.
Yes. This paper is attempting to provide an infrastructure using which one can do dynamic optimization of his/her code during runtime to improve program performance. Also, the authors are trying to minimize the associated overheads while maintaining flexibility, ease of use and transparency.

This is important considering the way modern software is designed and coded – there is increasing use of techniques like dynamic class loading, shared libraries, runtime binding, etc. Due to these runtime techniques, a 'static' compiler can not analyze the whole program. At best, it can be guided by profile information, but this works well only for programs that have relatively constant dynamic behavior.

Some benefits of shifting some of the optimizations to runtime are:

- Optimizations can be performed adaptively, depending on the runtime conditions
- Architecture-specific optimizations can be performed at runtime on a 'generic' binary, so that the 'static' compiler need not bother about the machine-version-specific details
- A dynamic optimizer can optimize code which was linked dynamically

Are the proposed approaches valid? Describe its strength and weakness.
The proposed approached are valid, but have a significant overhead – despite a lot of effort to minimize the overhead.

Strengths
Several approaches have been taken to reduce overhead and increase efficiency:

- Optimizations are restricted to linear streams of code, so that simpler and hence faster optimizations can be used
- Different levels of details are used to represent the instructions, so that space efficiency is maximized
- Private code caches are used for individual threads, so that the overhead of inter-thread synchronization is avoided
- To avoid hash table look-up overhead while performing indirect jumps, runtime profiling techniques are used to record the ‘hot’ targets

The dynamic optimizer is transparent, so it does not conflict with the application program’s data. It is easy to use – the API exported by DynamoRIO is quite ‘rich’ and it hides all the gory details of the instruction representation, data structures, trace gathering, etc. from the user. Thus, the user can concentrate on writing trace analysis and optimization routines. The infrastructure is quite flexible and a variety of optimizations can be performed using DynamoRIO. It also supports adaptive re-optimization of code already present in the trace cache and building of custom traces based on user's criteria. Also, since the optimizer is quite general in nature, it can be used for other applications like
profiling, on-the-fly instruction decompression (can be applied to VLIW instructions) and decryption (can be used for XOM-like machines), etc.

**Weaknesses**

The main weakness is that the whole process of dynamic optimization incurs a lot of overhead and results in overall slowdown of the programs being executed – the opposite of what is intended! The approaches work well mostly for FP programs, but not so well for integer programs.

**Do the results support the conclusions? Explain.**

The main conclusions are that DynamoRIO is flexible yet efficient and transparent. The results do support the flexibility and transparency parts of the conclusion – but there is still a lot of room for improving the efficiency. This is especially true for integer programs, which on an average suffer slowdown. However, the optimizations that they have demonstrated in the paper do produce some speed up as compared to the base DynamoRIO infrastructure.

**Describe the potential future works?**

There are a lot of things that could be done:

- They try to perform optimizations throughout the program runtime. It might be better to perform optimizations only when the program is in stable phases. This will reduce the overheads and increase the efficiency – potentially giving better speedup across the board.
- All the ‘levels’ of instruction representation are probably not required; some can be eliminated (like level 1) to gain some benefit.
- The ‘hot’ target list is not updated once a certain threshold has been reached. Updating this list with the current ‘hot’ targets will surely give some performance improvement – provided that the replacement policy does not incur too much overhead.
- DynamoRIO suffers from more costly indirect branch misprediction penalty than the native application. They can use more conservative approached to predicting indirect branches to avoid the high penalty.
- Applications of DynamoRIO to other purposes (like the ones mentioned above) can be explored. I think that applications to instruction decompression and decryption are quite feasible.